Guia do desenvolvedor do SDK versão 3

AWS SDK for JavaScript

AWS SDK for JavaScript: Guia do desenvolvedor do SDK versão 3

Table of Contents

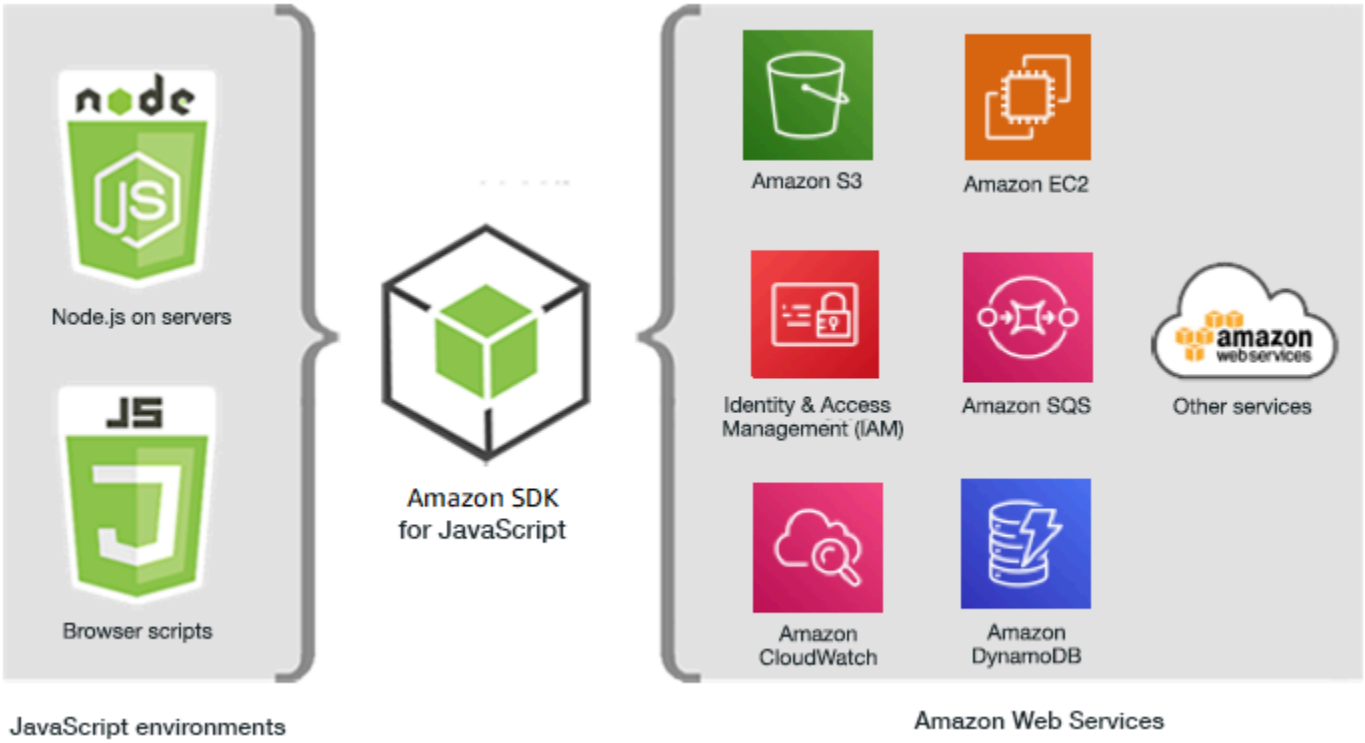
O Guia de referência da API do AWS SDK for JavaScript V3 descreve em detalhes todas as operações da API para o AWS SDK for JavaScript versão 3 (V3).

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.

O que é o AWS SDK for JavaScript?

Bem-vindo ao Guia do AWS SDK for JavaScript desenvolvedor. Este guia fornece informações gerais sobre como instalar e configurar o AWS SDK for JavaScript. Ele também mostra exemplos e tutoriais de execução de vários AWS serviços usando AWS SDK for JavaScript o.

O Guia de referência da API AWS SDK for JavaScript v3 fornece uma JavaScript API para AWS serviços. Você pode usar a JavaScript API para criar bibliotecas ou aplicativos para o Node.js ou para o navegador.



Começar a usar o SDK

Se você estiver pronto para começar a usar o SDK, siga os exemplos em. Conceitos básicos

Consulte Configure o SDK para JavaScript para configurar seu ambiente de desenvolvimento.

Se você estiver usando a versão 2.x do SDK para JavaScript, consulte Migrar para a v3 para obter orientações específicas.

Se você está procurando exemplos de código para Serviços da AWS, consulteSDK para exemplos de JavaScript código (v3).

Manutenção e suporte para as versões principais do SDK

Para obter informações sobre manutenção e suporte para versões principais do SDK e suas dependências subjacentes, consulte o seguinte no Guia de referência de AWS SDKs e ferramentas:

- AWS Política de manutenção de SDKs e ferramentas
- AWS Matriz de suporte de versões de SDKs e ferramentas

Usar o SDK com o Node.js

O Node.js é um tempo de execução multiplataforma para executar aplicativos do lado do servidor JavaScript . Você pode configurar o Node.js em uma instância do Amazon Elastic Compute Cloud (Amazon EC2) para executar em um servidor. Você também pode usar o Node.js para escrever AWS Lambda funções sob demanda.

O uso do SDK para Node.js é diferente da forma como você o usa JavaScript em um navegador da Web. A diferença refere-se à maneira como você carrega o SDK e obtém as credenciais necessárias para acessar serviços da web específicos. Quando o uso de determinadas APIs diferir entre o Node.js e o navegador, essas diferenças serão destacadas.

Usando o SDK com AWS Cloud9

Você também pode desenvolver aplicativos Node.js usando o SDK para JavaScript no AWS Cloud9 IDE. Para obter mais informações sobre como usar AWS Cloud9 com o SDK para JavaScript, consulteUse AWS Cloud9 com o AWS SDK for JavaScript.

Usando o SDK com AWS Amplify

Para aplicativos web, móveis e híbridos baseados em navegador, você também pode usar a AWS Amplify biblioteca em. GitHub Ele estende o SDK para JavaScript, fornecendo uma interface declarativa.

> **Note**
>
> Frameworks como o Amplify podem não oferecer o mesmo suporte de navegador que o SDK para. JavaScript Consulte a documentação da estrutura para obter detalhes.

Uso do SDK com navegadores da web

Todos os principais navegadores da web suportam a execução de JavaScript. JavaScript o código executado em um navegador da Web geralmente é chamado de lado do cliente JavaScript.

Para obter uma lista de navegadores compatíveis com o AWS SDK for JavaScript, consulteNavegadores da Web compatíveis.

Usar o SDK para JavaScript em um navegador da Web é diferente da maneira como você o usa para Node.js. A diferença refere-se à maneira como você carrega o SDK e obtém as credenciais necessárias para acessar serviços da web específicos. Quando o uso de determinadas APIs diferir entre o Node.js e o navegador, essas diferenças serão destacadas.

Uso dos navegadores na V3

A V3 permite que você agrupe e inclua no navegador somente o SDK para os JavaScript arquivos necessários, reduzindo a sobrecarga.

Para usar a V3 do SDK JavaScript em suas páginas HTML, você deve agrupar os módulos de cliente necessários e todas as JavaScript funções necessárias em um único JavaScript arquivo usando o Webpack e adicioná-lo em uma tag <head> de script nas suas páginas HTML. Por exemplo: .

```
<script src="./main.js"></script>
```

> **Note**
>
> Para obter mais informações sobre o Webpack, consulte Agrupe aplicativos com o webpack.

Para usar a V2 do SDK JavaScript, você adiciona uma tag de script que aponta para a versão mais recente do SDK V2. Para obter mais informações, consulte o exemplo no Guia do AWS SDK for JavaScript desenvolvedor v2.

Casos de uso comuns

Usar o SDK para scripts JavaScript em navegadores possibilita a realização de vários casos de uso convincentes. Aqui estão várias ideias de coisas que você pode criar em um aplicativo de navegador usando o SDK JavaScript para acessar vários serviços da Web.

- Crie um console personalizado para AWS serviços nos quais você acessa e combina recursos entre regiões e serviços para melhor atender às suas necessidades organizacionais ou de projeto.
- Use o Amazon Cognito Identity para habilitar o acesso do usuário autenticado aos aplicativos de navegador e sites, incluindo o uso de autenticação de terceiros pelo Facebook e outros.
- Use o Amazon Kinesis para processar clickstreams ou outros dados de marketing em tempo real.
- Use o Amazon DynamoDB para persistência de dados sem servidor, como preferências de usuários individuais quanto a visitantes do site ou usuários de aplicativos.
- Use AWS Lambda para encapsular a lógica proprietária que você pode invocar a partir de scripts de navegador sem baixar e revelar sua propriedade intelectual aos usuários.

Sobre os exemplos

Você pode procurar JavaScript exemplos no SDK no AWS Code Example Repository.

Recursos

Além desse guia, os seguintes recursos on-line estão disponíveis para o SDK para JavaScript desenvolvedores:

- AWS SDK for JavaScript Guia de referência da API V3
- AWS Guia de referência de SDKs e ferramentas: contém configurações, recursos e outros conceitos fundamentais comuns entre os AWS SDKs.
- JavaScript Blog do desenvolvedor
- AWS JavaScript Fórum
- JavaScript exemplos no Catálogo AWS de códigos
- AWS Repositório de exemplos de código
- Canal Gitter
- Stack Overflow
- Perguntas do Stack Overflow com a tag AWS -sdk-js
- GitHub
  - Fonte do SDK
  - Fonte de documentação

Comece com o AWS SDK for JavaScript

O AWS SDK for JavaScript fornece acesso aos serviços da Web em um navegador ou em um ambiente Node.js. Esta seção contém exercícios de introdução que mostram como trabalhar com o SDK JavaScript em cada um desses JavaScript ambientes.

> **Note**
>
> Você pode desenvolver aplicativos Node.js e, JavaScript para aplicativos baseados em navegador, usando o SDK do IDE JavaScript . AWS Cloud9 Para obter um exemplo de como usar AWS Cloud9 para o desenvolvimento do Node.js, consulteUse AWS Cloud9 com o AWS SDK for JavaScript.

Tópicos

Autenticação do SDK com AWS

Você deve estabelecer como seu código é autenticado AWS ao desenvolver com Serviços da AWS. Você pode configurar o acesso programático aos AWS recursos de maneiras diferentes, dependendo do ambiente e do AWS acesso disponível para você.

Para escolher seu método de autenticação e configurá-lo para o SDK, consulte Autenticação e acesso no Guia de referência de ferramentas e SDKs da AWS .

Recomendamos que novos usuários que estejam se desenvolvendo localmente e que não recebam um método de autenticação do empregador se configurem AWS IAM Identity Center. Esse método inclui a instalação do AWS CLI para facilitar a configuração e entrar regularmente no portal de AWS acesso. Se você escolher esse método, seu ambiente deverá conter os seguintes elementos depois de concluir o procedimento de autenticação do IAM Identity Center no Guia de referência de ferramentas e SDKs da AWS :

- O AWS CLI, que você usa para iniciar uma sessão do portal de AWS acesso antes de executar seu aplicativo.

- Um arquivo AWS`config` compartilhado com um perfil de `[default]` com um conjunto de valores de configuração que podem ser referenciados a partir do SDK. Para encontrar a localização desse arquivo, consulte Localização dos arquivos compartilhados no Guia de referência de ferramentas e SDKs da AWS .
- O arquivo `config` compartilhado define a configuração do `region`. Isso define o padrão Região da AWS que o SDK usa para AWS solicitações. Essa região é usada para solicitações de serviço do SDK que não são fornecidas com uma Região específica para uso.
- O SDK usa a configuração do provedor do token de SSO do perfil para adquirir credenciais antes de enviar solicitações para a AWS. O `sso_role_name` valor, que é uma função do IAM conectada a um conjunto de permissões do IAM Identity Center, permite acesso ao Serviços da AWS usado em seu aplicativo.

  O arquivo `config` de amostra a seguir mostra um perfil padrão configurado com o provedor de token de SSO. A configuração `sso_session` do perfil se refere à seção do `sso-session`. A `sso-session` seção contém configurações para iniciar uma sessão do portal de AWS acesso.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

A AWS SDK for JavaScript v3 não precisa que pacotes adicionais (como `SSO` e`SSOOIDC`) sejam adicionados ao seu aplicativo para usar a autenticação do IAM Identity Center.

Para obter detalhes sobre o uso explícito desse provedor de credenciais, consulte `fromSSO()` no site do npm (gerenciador de pacotes Node.js).

Iniciar uma sessão do portal de AWS acesso

Antes de executar um aplicativo que acessa Serviços da AWS, você precisa de uma sessão ativa do portal de AWS acesso para que o SDK use a autenticação do IAM Identity Center para resolver as

credenciais. Dependendo da duração da sessão configurada, o seu acesso acabará expirando e o SDK encontrará um erro de autenticação. Para entrar no portal de AWS acesso, execute o seguinte comando no AWS CLI.

```
aws sso login
```

Se você seguiu as orientações e tem um perfil padrão configurado, não precisará chamar o comando com uma opção de `--profile`. Se a configuração do provedor de token de SSO estiver usando um perfil nomeado, o comando será `aws sso login --profile named-profile`.

Para testar opcionalmente se você já tem uma sessão ativa, execute o AWS CLI comando a seguir.

```
aws sts get-caller-identity
```

Se a sua sessão estiver ativa, a resposta a este comando relata a conta do IAM Identity Center e o conjunto de permissões configurados no arquivo `config` compartilhado.

> **Note**
>
> Se você já tiver uma sessão ativa do portal de AWS acesso e executá-la`aws sso login`, não será necessário fornecer credenciais.
> O processo de login pode solicitar que você permita o AWS CLI acesso aos seus dados. Como o AWS CLI é criado com base no SDK para Python, as mensagens de permissão podem conter variações do `botocore` nome.

Mais informações de autenticação

Os usuários humanos, também conhecidos como identidades humanas, são as pessoas, os administradores, os desenvolvedores, os operadores e os consumidores de suas aplicações. Eles devem ter uma identidade para acessar seus AWS ambientes e aplicativos. Usuários humanos que são membros da sua organização (ou seja, você, o desenvolvedor) são conhecidos como identidades da força de trabalho.

Use credenciais temporárias ao acessar AWS. Você pode usar um provedor de identidade para seus usuários humanos para fornecer acesso federado às AWS contas assumindo funções que fornecem credenciais temporárias. Para gerenciamento de acesso centralizado, recomendamos que você use o AWS IAM Identity Center (IAM Identity Center) para gerenciar o acesso às suas contas e as permissões nessas contas. Para obter mais alternativas, consulte as informações a seguir.

- Para saber mais sobre as práticas recomendadas, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.
- Para criar AWS credenciais de curto prazo, consulte Credenciais de segurança temporárias no Guia do usuário do IAM.
- Para saber mais sobre outros provedores de credenciais AWS SDK for JavaScript V3, consulte Provedores de credenciais padronizados no Guia de referência de AWS SDKs e ferramentas.

Conceitos básicos sobre o Node.js

Este guia mostra como inicializar um pacote NPM, adicionar um cliente de serviço ao seu pacote e usar o JavaScript SDK para chamar uma ação de serviço.

O cenário

Crie um novo pacote NPM com um arquivo principal que faz o seguinte:

- Cria um bucket do Amazon Simple Storage Service
- Coloca um objeto no bucket do Amazon S3
- Lê o objeto no bucket do Amazon S3
- Confirma se o usuário deseja excluir recursos

Pré-requisitos

Antes de executar o exemplo, faça o seguinte:

- Configure a autenticação do SDK. Para ter mais informações, consulte Autenticação do SDK com AWS.
- Instale o Node.js.

Etapa 1: configurar a estrutura do pacote e instalar pacotes do cliente

Para configurar a estrutura do pacote e instalar pacotes do cliente:

1. Crie uma nova pasta `nodegetstarted` para conter o pacote.
2. Na linha de comando, navegue até a nova pasta.
3. Execute o seguinte comando para criar um arquivo `package.json` padrão:

```
npm init -y
```

4. Execute o comando a seguir para instalar o pacote de cliente do Amazon S3:

```
npm i @aws-sdk/client-s3
```

5. Adicione `"type": "module"` ao arquivo `package.json`. Isso faz com que o Node.js use a sintaxe moderna do ESM. O arquivo `package.json` final deverá ser semelhante ao seguinte:

```
{
  "name": "example-javascriptv3-get-started-node",
  "version": "1.0.0",
  "description": "This guide shows you how to initialize an NPM package, add a
 service client to your package, and use the JavaScript SDK to call a service
 action.",
  "main": "index.js",
  "scripts": {
"test": "vitest run **/*.unit.test.js"
  },
  "author": "Your Name",
  "license": "Apache-2.0",
  "dependencies": {
 "@aws-sdk/client-s3": "^3.420.0"
  },
  "type": "module"
}
```

Etapa 2: Adicionar as importações e o código SDK necessários

Adicione o código a seguir a um arquivo denominado `index.js` na pasta `nodegetstarted`.

```
// This is used for getting user input.
import { createInterface } from "readline/promises";

import {
  S3Client,
  PutObjectCommand,
  CreateBucketCommand,
```

```
  DeleteObjectCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  GetObjectCommand,
} from "@aws-sdk/client-s3";

export async function main() {
  // A region and credentials can be declared explicitly. For example
  // `new S3Client({ region: 'us-east-1', credentials: {...} })` would
  //initialize the client with those settings. However, the SDK will
  // use your local configuration and credentials if those properties
  // are not defined here.
  const s3Client = new S3Client({});

  // Create an Amazon S3 bucket. The epoch timestamp is appended
  // to the name to make it unique.
  const bucketName = `test-bucket-${Date.now()}`;
  await s3Client.send(
    new CreateBucketCommand({
      Bucket: bucketName,
    })
  );

  // Put an object into an Amazon S3 bucket.
  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "my-first-object.txt",
      Body: "Hello JavaScript SDK!",
    })
  );

  // Read the object.
  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucketName,
      Key: "my-first-object.txt",
    })
  );

  console.log(await Body.transformToString());

  // Confirm resource deletion.
  const prompt = createInterface({
```

```
    input: process.stdin,
    output: process.stdout,
  });

  const result = await prompt.question("Empty and delete bucket? (y/n) ");
  prompt.close();

  if (result === "y") {
    // Create an async iterator over lists of objects in a bucket.
    const paginator = paginateListObjectsV2(
      { client: s3Client },
      { Bucket: bucketName }
    );
    for await (const page of paginator) {
      const objects = page.Contents;
      if (objects) {
        // For every object in each page, delete it.
        for (const object of objects) {
          await s3Client.send(
            new DeleteObjectCommand({ Bucket: bucketName, Key: object.Key })
          );
        }
      }
    }

    // Once all the objects are gone, the bucket can be deleted.
    await s3Client.send(new DeleteBucketCommand({ Bucket: bucketName }));
  }
}

// Call a function if this file was run directly. This allows the file
// to be runnable without running on import.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

O código de exemplo pode ser encontrado aqui em GitHub.

Etapa 3: Executar o exemplo

> **Note**
>
> Lembre-se de fazer login! Se você estiver usando o IAM Identity Center para se autenticar, lembre-se de fazer login usando o AWS CLI `aws sso login` comando.

1. Executar `node index.js`.
2. Escolha se deseja esvaziar e excluir o bucket.
3. Se você não excluir o bucket, certifique-se de esvaziá-lo manualmente e excluí-lo mais tarde.

Conceitos básicos sobre o navegador

Esta seção mostra um exemplo que demonstra como executar a versão 3 (V3) do SDK JavaScript no navegador.

> **Note**
>
> A execução da V3 no navegador é um pouco diferente da versão 2 (V2). Para ter mais informações, consulte Uso dos navegadores na V3.

Para outros exemplos de uso (V3) do SDK para JavaScript, consulte. SDK para exemplos de JavaScript código (v3)

Este exemplo de aplicativo web mostra:

- Como acessar AWS serviços usando o Amazon Cognito para autenticação.
- Como ler uma lista de objetos em um bucket do Amazon Simple Storage Service (Amazon S3) usando AWS Identity and Access Management uma função (IAM).

> **Note**
>
> Este exemplo não é usado AWS IAM Identity Center para autenticação.

O cenário

O Amazon S3 é um serviço de armazenamento de objetos que oferece escalabilidade, disponibilidade de dados, segurança e performance líderes do setor. Você pode usar o Amazon S3 para armazenar dados como objetos em contêineres chamados buckets. Para obter mais informações sobre o Amazon S3, consulte o Guia do usuário da Amazon S3.

Este exemplo mostra como configurar e executar um aplicativo web que assume um perfil do IAM para ler de um bucket do Amazon S3. O exemplo usa a biblioteca front-end React e as ferramentas front-end Vite para fornecer um ambiente de desenvolvimento. JavaScript O aplicativo web usa um pool de identidade do Amazon Cognito para fornecer as credenciais necessárias para acessar os serviços. AWS O exemplo de código incluído demonstra os padrões básicos para carregar e usar o SDK JavaScript em aplicativos web.

Etapa 1: Criar um banco de identidades e um perfil do IAM do Amazon Cognito

Neste exercício, você cria e usa um banco de identidades do Amazon Cognito para fornecer acesso não autenticado ao aplicativo web do serviço Amazon S3. A criação de um grupo de identidades também cria uma função AWS Identity and Access Management (IAM) para oferecer suporte a usuários convidados não autenticados. Neste exemplo, vamos trabalhar apenas com a função de usuário não autenticado para manter o enfoque na tarefa. Você poderá integrar o suporte para um provedor de identidade e os usuários autenticados depois. Para obter mais informações sobre como adicionar um banco de identidades do Amazon Cognito, consulte Tutorial: criação de um banco de identidades no Guia do desenvolvedor do Amazon Cognito.

Para criar um banco de identidades e um perfil do IAM associado do Amazon Cognito

1. Faça login AWS Management Console e abra o console do Amazon Cognito em https://console.aws.amazon.com/cognito/.
2. No painel de navegação à esquerda, escolha Bancos de identidades.
3. Selecione Criar banco de identidades.
4. Em Configurar confiança do grupo de identidades, escolha Acesso de convidado para autenticação do usuário.
5. Em Configurar permissões, escolha Criar uma nova função do IAM e insira um nome (por exemplo, get StartedRole) no nome da função do IAM.

6. Em Configurar propriedades, insira um nome (por exemplo, get StartedPool) em Nome do grupo de identidades.
7. Em Revisar e criar, confirme as seleções que você fez para o novo banco de identidades. Selecione Editar para retornar ao assistente e alterar as configurações. Quando terminar, selecione Criar banco de identidades.
8. Observe o ID do grupo de identidades e a Região do banco de identidades recém-criado do Amazon Cognito. Você precisa desses valores para substituir *IDENTITY_POOL_ID* e *REGION* na Etapa 4: Configurar o código do navegador.

Depois de criar o banco de identidades do Amazon Cognito, você estará pronto para adicionar permissões do Amazon S3 necessárias para o aplicativo web.

Etapa 2: Adicionar uma política ao perfil do IAM criado

Para permitir o acesso a um bucket do Amazon S3 em seu aplicativo web, use a função IAM não autenticada (por exemplo, get StartedRole) criada para seu pool de identidade do Amazon Cognito (por exemplo, get). StartedPool Isso exige que você anexe uma política do IAM ao perfil. Para obter mais informações sobre como modificar os perfis do IAM, consulte Modificação de uma política de permissões de perfil no Guia do usuário do IAM.

Para adicionar uma política do Amazon S3 ao perfil do IAM associado a usuários não autenticados

1. Faça login AWS Management Console e abra o console do IAM em https://console.aws.amazon.com/iam/.
2. No painel de navegação à esquerda, escolha Roles.
3. Escolha o nome da função que você deseja modificar (por exemplo, obter StartedRole) e, em seguida, escolha a guia Permissões.
4. Escolha Adicionar permissões e depois Anexar políticas.
5. Na página Adicionar permissões para essa função, localize e marque a caixa de seleção do ReadOnlyAmazonS3 Access.

> **Note**
>
> Você pode usar esse processo para permitir o acesso a qualquer AWS serviço.

6. Escolha Add permissions (Adicionar permissões).

Depois de criar o banco de identidades do Amazon Cognito e adicionar permissões do Amazon S3 ao perfil do IAM para usuários não autenticados, você estará pronto para adicionar e configurar um bucket do Amazon S3.

Etapa 3: Adicionar um bucket e um objeto do Amazon S3

Nesta etapa, você adicionará um bucket e um objeto do Amazon S3 como exemplo. Você também poderá fazer o compartilhamento de recursos de origem cruzada (CORS) para o bucket. Para obter mais informações sobre como criar buckets e objetos do Amazon S3, consulte Conceitos básicos do Amazon S3 no Guia do usuário do Amazon S3.

Para adicionar um bucket e um objeto do Amazon S3 com CORS

1. Faça login no AWS Management Console e abra o console do Amazon S3 em https://console.aws.amazon.com/s3/.
2. No painel de navegação à esquerda, escolha Buckets e selecione Criar bucket.
3. Insira um nome de bucket que esteja em conformidade com as regras de nomenclatura de bucket (por exemplo, getstartedbucket) e escolha Criar bucket.
4. Escolha o bucket que você criou e, em seguida, escolha a guia Objetos. Em seguida, escolha Upload.
5. Em Files and folders (Arquivos e pastas), escolha Add files (Adicionar arquivos).
6. Escolha um arquivo para carregar e, em seguida, escolha Open (Abrir). Em seguida, escolha Carregar para concluir o carregamento do objeto no seu bucket.
7. Em seguida, escolha a guia Permissões do seu bucket e selecione Editar na seção Compartilhamento de recursos de origem cruzada (CORS). Insira o seguinte JSON:

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
```

```
  }
]
```

8. Escolha Salvar alterações.

Depois de adicionar um bucket do Amazon S3 e um objeto, você estará pronto para configurar o código do navegador.

Etapa 4: Configurar o código do navegador

O aplicativo de exemplo consiste em um aplicativo React de página única. Os arquivos desse exemplo podem ser encontrados aqui em GitHub.

Para configurar o aplicativo de exemplo

1. Instale o Node.js.
2. Na linha de comando, clone o Repositório de exemplos de código da AWS:

```
git clone --depth 1 https://github.com/awsdocs/aws-doc-sdk-examples.git
```

3. Navegue até o aplicativo de exemplo:

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

4. Execute o seguinte comando para instalar os pacotes necessários:

```
npm install
```

5. Em seguida, abra `src/App.tsx` em um editor de texto e conclua o seguinte:

   - Substitua *`YOUR_IDENTITY_POOL_ID`* pelo ID do banco de identidades do Amazon Cognito que você anotou em Etapa 1: Criar um banco de identidades e um perfil do IAM do Amazon Cognito.
   - Substitua o valor da região pela região atribuída ao seu bucket do Amazon S3 e ao banco de identidades do Amazon Cognito. Observe que as regiões de ambos os serviços devem ser as mesmas (por exemplo, us-east-2).
   - Substitua *`bucket-name`* pelo nome do bucket criado em Etapa 3: Adicionar um bucket e um objeto do Amazon S3.

Depois de substituir o texto, salve o arquivo `App.tsx`. Agora, você está pronto para executar o aplicativo web.

Etapa 5: Executar o exemplo

Para executar o aplicativo de exemplo

1. Na linha de comando, navegue até o aplicativo de exemplo:

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

2. Na linha de comando, execute o seguinte comando:

```
npm run dev
```

   O ambiente de desenvolvimento do Vite será executado com a seguinte mensagem:

```
VITE v4.3.9  ready in 280 ms

#  Local:   http://localhost:5173/
#  Network: use --host to expose
#  press h to show help
```

3. No navegador da Web, acesse o URL mostrado acima (por exemplo, http://localhost:5173). O aplicativo de exemplo mostrará uma lista de nomes de arquivos de objetos em seu bucket do Amazon S3.

Limpeza

Para limpar os recursos que foram criados durante este tutorial, faça o seguinte:

- No console do Amazon S3, exclua todos os objetos e buckets criados (por exemplo, getstartedbucket).
- No console do IAM, exclua o nome da função (por exemplo, get StartedRole).
- No console do Amazon Cognito, exclua o nome do grupo de identidades (por exemplo, get StartedPool).

Configure o SDK para JavaScript

Os tópicos desta seção explicam como instalar e carregar o SDK para JavaScript que você possa acessar os serviços web compatíveis com o SDK.

> **Note**
>
> Os desenvolvedores do React Native devem usar AWS Amplify para criar novos projetos no AWS. Consulte o aws-sdk-react-nativearquivo para obter detalhes.

Tópicos

Pré-requisitos

Instale o Node.js nos seus servidores, se ele ainda não estiver instalado.

Tópicos

Configurar um ambiente AWS Node.js

Para configurar um ambiente AWS Node.js no qual você possa executar seu aplicativo, use qualquer um dos seguintes métodos:

- Escolha uma imagem de máquina da Amazon (AMI) com o Node.js pré-instalado. Em seguida, crie uma instância do Amazon EC2 usando essa AMI. Ao criar sua instância do Amazon EC2, selecione sua AMI no AWS Marketplace. Pesquise AWS Marketplace por Node.js e escolha uma opção de AMI que inclua uma versão pré-instalada do Node.js (32 bits ou 64 bits).

- Crie uma instância do Amazon EC2 e instale o Node.js. Para obter mais informações sobre como instalar o Node.js em uma instância do Amazon Linux, consulte Configuração do Node.js em uma instância do Amazon EC2.
- Crie um ambiente sem servidor usando AWS Lambda para executar o Node.js como uma função Lambda. Para obter mais informações sobre como usar o Node.js em uma função do Lambda, consulte Modelo de programação (Node.js) no Guia do desenvolvedor do AWS Lambda .
- Implante seu aplicativo Node.js em AWS Elastic Beanstalk. Para obter mais informações sobre como usar o Node.js com Elastic Beanstalk, consulte Implantar aplicativos do Node.js no AWS Elastic Beanstalk no Guia do Desenvolvedor do AWS Elastic Beanstalk .
- Crie um servidor de aplicativos Node.js usando AWS OpsWorks o. Para obter mais informações sobre como usar o Node.js com AWS OpsWorks, consulte Como criar sua primeira pilha de Node.js no Guia do AWS OpsWorks usuário.

Navegadores da Web compatíveis

O AWS SDK for JavaScript suporta todos os navegadores da web modernos.

Na versão 3.183.0 ou posterior, o SDK JavaScript usa artefatos ES2020, que oferecem suporte às seguintes versões mínimas.

Navegador	Version (Versão)
Google Chrome	80,0 ou mais
Mozilla Firefox	80,0 ou mais
Opera	Mais de 63,0
Microsoft Edge	80,0 ou mais
Apple Safari	14,1+
Internet da Samsung	12.0+

Na versão 3.182.0 ou anterior, o SDK JavaScript usa artefatos ES5, que oferecem suporte às seguintes versões mínimas.

Navegador	Version (Versão)
Google Chrome	Mais de 49,0
Mozilla Firefox	45,0+
Opera	Mais de 36,0
Microsoft Edge	12.0+
Windows Internet Explorer	N/D
Apple Safari	9.0+
Navegador do Android	Mais de 76,0
UC Browser	12,12 ou mais
Internet da Samsung	5.0+

> **Note**
>
> Estruturas como essa AWS Amplify podem não oferecer o mesmo suporte de navegador que o SDK para. JavaScript Consulte a Documentação do AWS Amplify para obter detalhes.

Instale o SDK para JavaScript

Nem todos os serviços estão imediatamente disponíveis no SDK ou em todas as AWS regiões.

Para instalar um serviço AWS SDK for JavaScript usando o npm, o gerenciador de pacotes Node.js, digite o seguinte comando no prompt de comando, em que *SERVICE* é o nome de um serviço, comos3.

```
npm install @aws-sdk/client-SERVICE
```

Para obter uma lista completa dos pacotes do cliente de AWS SDK for JavaScript serviço, consulte o guia de referência AWS SDK for JavaScript da API.

Carregue o SDK para JavaScript

Depois de instalar o SDK, você pode carregar um pacote de cliente no aplicativo do seu nó usando `import`. Por exemplo, para carregar o cliente Amazon S3 e o comando Amazon ListBucketsS3, use o seguinte.

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
```

Configure o SDK para JavaScript

Antes de usar o SDK para JavaScript invocar serviços web usando a API, você deve configurar o SDK. No mínimo, você deve configurar:

- A AWS região na qual você solicitará serviços
- Como seu código é autenticado com AWS

Além dessas configurações, talvez você também precise configurar permissões para os recursos da AWS . Por exemplo, limite o acesso a um bucket do Amazon S3 ou restrinja uma tabela do Amazon DynamoDB para acesso somente leitura.

O Guia de referência de AWS SDKs e ferramentas também contém configurações, recursos e outros conceitos fundamentais comuns entre muitos dos AWS SDKs.

Os tópicos desta seção descrevem as formas de configurar o SDK JavaScript para Node.js e JavaScript executá-lo em um navegador da Web.

Tópicos

Configuração por serviço

Você pode configurar o SDK passando informações de configuração para um objeto de serviço.

A configuração em nível de serviço fornece controle significativo sobre serviços individuais, permitindo que você atualize a configuração de objetos de serviço individuais quando suas necessidades variam da configuração padrão.

> **Note**
>
> Na versão 2.x, a configuração do AWS SDK for JavaScript serviço pode ser passada para construtores de clientes individuais. No entanto, essas configurações primeiro serão mescladas automaticamente em uma cópia da configuração global do SDK: `AWS.config`. Além disso, a chamada de `AWS.config.update({/* params */})` somente atualizou a configuração para clientes de serviço instanciados depois que a chamada de atualização foi feita, e não para clientes existentes.
> Esse comportamento era uma fonte frequente de confusão e dificultava a adição de configuração ao objeto global que afeta apenas um subconjunto de clientes de serviço de forma compatível com versões futuras. Na versão 3, não há mais uma configuração global gerenciada pelo SDK. A configuração deve ser transmitida para cada cliente de serviço instanciado. Ainda é possível compartilhar a mesma configuração entre vários clientes, mas essa configuração não será automaticamente mesclada com um estado global.

Definir configuração por serviço

Cada serviço que você usa no SDK JavaScript é acessado por meio de um objeto de serviço que faz parte da API desse serviço. Por exemplo, para acessar o serviço Amazon S3, você cria o objeto de serviço Amazon S3. Especifique as definições de configuração específicas de um serviço como parte do construtor desse objeto de serviço.

Por exemplo, se você precisar acessar objetos do Amazon EC2 em várias AWS regiões, crie um objeto de serviço do Amazon EC2 para cada região e, em seguida, defina a configuração regional de cada objeto de serviço adequadamente.

```
var ec2_regionA = new EC2({region: 'ap-southeast-2', maxAttempts: 15});
var ec2_regionB = new EC2({region: 'us-west-2', maxAttempts: 15});
```

Defina a AWS região

Uma AWS região é um conjunto nomeado de AWS recursos na mesma área geográfica. Um exemplo de uma Região é `us-east-1`, que é a Região Leste dos EUA (Norte da Virgínia). Você especifica uma região ao criar um cliente de serviço no SDK para JavaScript que o SDK acesse o serviço nessa região. Alguns serviços só estão disponíveis em regiões específicas.

O SDK do JavaScript não seleciona uma região por padrão. No entanto, você pode definir a AWS Região usando uma variável de ambiente ou um `config` arquivo de configuração compartilhado.

Em um construtor de classes do cliente

Ao instanciar um objeto de serviço, você pode especificar a AWS região desse recurso como parte do construtor da classe cliente, conforme mostrado aqui.

```
const s3Client = new S3.S3Client({region: 'us-west-2'});
```

Use uma variável de ambiente

Defina a região usando a variável de ambiente `AWS_REGION`. Se você definir essa variável, o SDK para a JavaScript lê e a usa.

Use um arquivo de configuração compartilhado

Assim como o arquivo de credenciais compartilhado permite armazenar credenciais para uso pelo SDK, você pode manter sua AWS região e outras configurações em um arquivo compartilhado com o nome `config` do SDK a ser usado. Se a variável de `AWS_SDK_LOAD_CONFIG` ambiente for definida como um valor verdadeiro, o SDK JavaScript pesquisará automaticamente um `config` arquivo quando ele for carregado. Onde você salva o arquivo `config` depende do sistema operacional:

- Usuários de Linux, macOS ou Unix: `~/.aws/config`
- Usuários do Windows: `C:\Users\USER_NAME\.aws\config`

Se não tiver um arquivo `config` compartilhado, você poderá criar um no diretório designado. No exemplo a seguir, o arquivo `config` define a região e o formato de saída.

```
[default]
   region=us-west-2
   output=json
```

Para obter mais informações sobre o uso de arquivos `config` e `credentials` compartilhados, consulte Arquivos compartilhados de configuração e credenciais no Guia de referência de ferramentas e SDKs da AWS .

Ordem de precedência para definir a região

A ordem de precedência de definição da região é a seguinte:

1. Se uma região for passada para um construtor de classe de cliente, essa região será usada.
2. Se uma região for definida na variável de ambiente, essa região será usada.
3. Caso contrário, a região definida no arquivo de configuração compartilhado será usada.

Definir credenciais

AWS usa credenciais para identificar quem está ligando para os serviços e se o acesso aos recursos solicitados é permitido.

Seja em execução em um navegador da Web ou em um servidor Node.js, seu JavaScript código deve obter credenciais válidas antes de poder acessar os serviços por meio da API. As credenciais podem ser definidas por serviço, passando credenciais diretamente para um objeto de serviço.

Há várias maneiras de definir credenciais que diferem entre o Node.js e JavaScript nos navegadores da Web. Os tópicos nesta seção descrevem como definir credenciais em Node.js ou navegadores da web. Em cada caso, as opções são apresentadas na ordem recomendada.

Melhores práticas para credenciais

A definição de credenciais apropriada garante que o aplicativo ou o script de navegador possa acessar os serviços e os recursos necessários ao mesmo tempo que minimiza a exposição a problemas de segurança que possam afetar aplicativos de missão crítica ou comprometer dados confidenciais.

Um princípio importante a ser aplicado durante a definição de credenciais é sempre conceder o menor privilégio necessário para a tarefa. É mais seguro fornecer permissões mínimas nos recursos e adicionar mais permissões adicionais conforme necessário, em vez de fornecer permissões que excedam o menor privilégio e, dessa forma, precisar corrigir problemas de segurança que possam ser descobertos depois. Por exemplo, a menos que você precise ler e gravar recursos individuais, como objetos em um bucket do Amazon S3 ou uma tabela do DynamoDB, defina essas permissões como somente leitura.

Para obter mais informações sobre como conceder o privilégio mínimo, consulte a seção Conceder privilégio mínimo do tópico Melhores práticas no Guia do usuário do IAM.

Tópicos

Definir credenciais em Node.js

Recomendamos que novos usuários que estejam se desenvolvendo localmente e que não recebam um método de autenticação do empregador se configurem AWS IAM Identity Center. Para ter mais informações, consulte Autenticação do SDK com AWS.

Há várias maneiras em Node.js de fornecer as credenciais para o SDK. Algumas dessas são mais seguras e outras oferecem mais comodidade durante o desenvolvimento de aplicativos. Ao obter credenciais em Node.js, tome cuidado ao confiar em mais de uma origem, como uma variável de ambiente e um arquivo JSON carregado. Altere as permissões em que o código é executado sem perceber a alteração que aconteceu.

AWS SDK for JavaScript A V3 fornece uma cadeia de provedores de credenciais padrão no Node.js, portanto, você não precisa fornecer um provedor de credenciais explicitamente. A cadeia de fornecedores de credenciais padrão tenta resolver as credenciais de várias fontes diferentes em uma determinada precedência, até que uma credencial seja retornada de uma das fontes. Você pode encontrar a cadeia de fornecedores de credenciais do SDK for JavaScript V3 aqui.

Cadeia de provedores de credenciais

Todos os SDKs têm uma série de locais (ou fontes) que eles verificam para encontrar credenciais válidas para usar para fazer uma solicitação a um AWS service (Serviço da AWS). Depois que as credenciais válidas são encontradas, a pesquisa é interrompida. Essa busca sistemática é chamada de cadeia de provedores de credenciais padrão.

Para cada etapa da cadeia, há maneiras diferentes de definir os valores. A definição de valores diretamente no código sempre tem precedência, seguida pela configuração como variáveis de ambiente e, em seguida, no AWS `config` arquivo compartilhado. Para obter mais informações, consulte Precedência das configurações no Guia de referência de ferramentas e SDKs da AWS .

O Guia de referência de AWS SDKs e ferramentas tem informações sobre as configurações do SDK usadas por todos os AWS SDKs e pelo. AWS CLI Para saber mais sobre como configurar o SDK por meio do AWS `config` arquivo compartilhado, consulte Arquivos de configuração e credenciais

compartilhados. Para saber mais sobre como configurar o SDK por meio da definição de variáveis de ambiente, consulte Suporte a variáveis de ambiente.

Para se autenticar AWS, ele AWS SDK for JavaScript verifica os provedores de credenciais na ordem listada na tabela a seguir.

AWS SDK for JavaScrip t Método do provedor de credenciais de referência da API por precedência	Fornecedor(es) de credenciais disponíveis	AWS Guia de referência de SDKs e ferramentas
`fromEnv()`	AWS chaves de acesso a partir de variáveis de ambiente	AWS chaves de acesso
`fromSSO()`	AWS IAM Identity Center. Neste guia, consulte Autentica ção do SDK com AWS.	Fornecedor de credenciais do IAM Identity Center
`fromIni()`	AWS chaves de acesso de `credentials` arquivos compartilhados `config` e compartilhados	AWS chaves de acesso
	Provedor de entidades confiável (como `AWS_ROLE_ ARN`)	Assumir um perfil do IAM
	Token de identidade da Web de AWS Security Token Service (AWS STS)	Federar com identidade da Web ou OpenID Connect
	Credencias do Amazon Elastic Container Service (Amazon ECS)	Provedor de credenciais de contêiner
	Credenciais do perfil de instância do Amazon Elastic Compute Cloud (Amazon	Provedor de credenciais do IMDS

AWS SDK for JavaScrip t Método do provedor de credenciais de referência da API por precedência	Fornecedor(es) de credenciais disponíveis	AWS Guia de referência de SDKs e ferramentas
	EC2) (provedor de credenciais do IMDS)	
	Provedor de credenciais de processo	Provedor de credenciais de processo
	AWS IAM Identity Center credenciais	Fornecedor de credenciais do IAM Identity Center
`fromProcess()`	Provedor de credenciais de processo	Provedor de credenciais de processo
`fromTokenFile()`	Token de identidade da Web de AWS Security Token Service (AWS STS)	Federar com identidade da Web ou OpenID Connect
`fromContainerMetadata()`	Credencias do Amazon Elastic Container Service (Amazon ECS)	Provedor de credenciais de contêiner
`fromInstanceMetadata()`	Credenciais do perfil de instância do Amazon Elastic Compute Cloud (Amazon EC2) (provedor de credenciais do IMDS)	Provedor de credenciais do IMDS

Se você seguiu a abordagem recomendada para novos usuários começarem, configurou a autenticação do AWS IAM Identity Center durante a Autenticação do SDK com AWS do tópico Conceitos básicos. Outros métodos de autenticação são úteis para situações diferentes. Para evitar riscos de segurança, recomendamos sempre usar credenciais de curto prazo. Para outros procedimentos de método de autenticação, consulte Autenticação e acesso no Guia de referência de ferramentas e SDKs da AWS .

Os tópicos nesta seção descrevem como carregar credenciais em Node.js.

Tópicos

Carregue credenciais em Node.js a partir de funções do IAM para o Amazon EC2

Se executar o aplicativo Node.js em uma instância do Amazon EC2, você poderá aproveitar perfis do IAM para o Amazon EC2 fornecer credenciais automaticamente para a instância. Se você configurar a instância para usar perfis do IAM, o SDK selecionará automaticamente as credenciais do IAM do aplicativo, eliminando a necessidade de fornecer credenciais manualmente.

Para obter mais informações sobre como adicionar perfis do IAM a uma instância do Amazon EC2, consulte perfis do IAM para o Amazon EC2.

Carregar credenciais para uma função Lambda do Node.js

Ao criar uma AWS Lambda função, você deve criar uma função especial do IAM que tenha permissão para executar a função. Essa função é chamada de função de execução. Ao configurar uma função do Lambda, você deve especificar o perfil do IAM que você criou como o perfil do execução correspondente.

A função de execução fornece a função do Lambda com as credenciais de que precisa para executar e invocar outros serviços da Web. Dessa maneira, você não precisa fornecer credenciais para o código Node.js gravado em uma função do Lambda.

Para obter mais informações sobre como criar uma função de execução do Lambda, consulte Gerenciar permissões: usar um perfil do IAM (função de execução) no Guia do desenvolvedor do AWS Lambda .

Definir credenciais em um navegador da web

Há várias maneiras de fornecer as credenciais para o SDK de scripts de navegador. Algumas dessas são mais seguras e outras oferecem mais comodidade durante o desenvolvimento de scripts.

Aqui estão as maneiras como é possível fornecer as credenciais em ordem de recomendação:

1. Usar o Amazon Cognito Identity para autenticar usuários e fornecer credenciais

2. Usar identidade federada da web

> **Warning**
>
> Não recomendamos codificar suas AWS credenciais em seus scripts. A codificação rígida de credenciais oferece um risco de expor o ID de chave de acesso e a chave de acesso secreta.

Tópicos

- Use o Amazon Cognito Identity para autenticar usuários

Use o Amazon Cognito Identity para autenticar usuários

A forma recomendada de obter AWS credenciais para os scripts do seu navegador é usar o cliente de credenciais do Amazon Cognito Identity. `CognitoIdentityClient` O Amazon Cognito permite a autenticação de usuários por meio de provedores de identidade terceirizados.

Para usar o Amazon Cognito Identity, você deve primeiro criar um banco de identidades no console do Amazon Cognito. Um grupo de identidades representa o grupo de identidades fornecido pelo aplicativo para os usuários. As identidades atribuídas a usuários identificam com exclusividade cada conta de usuário. As identidades do Amazon Cognito não são credenciais. Eles são trocados por credenciais usando o suporte à federação de identidade da web em AWS Security Token Service ()AWS STS.

O Amazon Cognito ajuda a gerenciar a abstração de identidades entre vários provedores de identidade. A identidade carregada acaba sendo trocada por credenciais no AWS STS.

Configurar o objeto de credenciais do Amazon Cognito Identity

Se você ainda não tiver criado um, crie um banco de identidades para usar com os scripts do navegador no console do Amazon Cognito antes de configurar o cliente do Amazon Cognito. Crie e associe os perfis do IAM autenticados e não autenticados para o banco de identidades. Para obter mais informações, consulte Tutorial: criação de um banco de identidades no Guia do desenvolvedor do Amazon Cognito.

Usuários não autenticados não têm a identidade verificada, tornando esse perfil apropriado para usuários convidados de seu aplicativo ou nos casos em que não importa se os usuários têm

suas identidades verificadas. Os usuários autenticados fazem login no aplicativo por meio de um provedor de identidade de terceiros que verifica as identidades. Certifique-se de definir o escopo das permissões dos recursos de forma apropriada para que você não conceda acesso a eles a partir de usuários não autenticados.

Depois de configurar um banco de identidades, use o método `fromCognitoIdentityPool` do `@aws-sdk/credential-providers` para recuperar as credenciais do banco de identidades. No exemplo a seguir de criação de um cliente do Amazon S3, substitua *`AWS_REGION`* pela região e *`IDENTITY_POOL_ID`* pelo ID do banco de identidades.

```
// Import required AWS SDK clients and command for Node.js
import {S3Client} from "@aws-sdk/client-s3";
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";

const REGION = AWS_REGION;

const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
 CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: {
            // Optional tokens, used for authenticated login.
        },
  })
});
```

A propriedade opcional `logins` é um mapa de nomes de provedor de identidade para os tokens de identidade para esses provedores. Como você obtém o token do seu provedor de identidade depende do provedor que usa. Por exemplo, se você estiver usando um grupo de usuários do Amazon Cognito como seu provedor de autenticação, poderá usar um método semelhante ao descrito abaixo.

```
// Get the Amazon Cognito ID token for the user. 'getToken()' below.
let idToken = getToken();
let COGNITO_ID = "COGNITO_ID"; // 'COGNITO_ID' has the format 'cognito-
idp.REGION.amazonaws.com/COGNITO_USER_POOL_ID'
let loginData = {
  [COGNITO_ID]: idToken,
```

```
};
const s3Client = new S3Client({
    region: REGION,
    credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
 CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: loginData
  })
});

// Strips the token ID from the URL after authentication.
window.getToken = function () {
  var idtoken = window.location.href;
  var idtoken1 = idtoken.split("=")[1];
  var idtoken2 = idtoken1.split("&")[0];
  var idtoken3 = idtoken2.split("&")[0];
  return idtoken3;
};
```

Trocar usuários não autenticados por usuários autenticados

O Amazon Cognito é compatível com usuários autenticados e não autenticados. Os usuários não autenticados receberão acesso aos recursos se eles não estiverem conectados a nenhum dos provedores de identidade. Esse nível de acesso é útil para exibir conteúdo para usuários antes de fazer login. Cada usuário não autenticado tem uma identidade exclusiva no Amazon Cognito, mesmo que não tenha feito login e sido autenticado individualmente.

Usuário não autenticado inicialmente

Os usuários normalmente começam com a função não autenticada para a qual você define a propriedade de credenciais do objeto de configuração sem uma propriedade `logins`. Neste caso, suas credenciais padrão podem parecer com o seguinte:

```
// Import the required AWS SDK for JavaScript v3 modules.
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";
// Set the default credentials.
const creds = fromCognitoIdentityPool({
  identityPoolId: 'IDENTITY_POOL_ID',
  clientConfig: { region: REGION } // Configure the underlying CognitoIdentityClient.
});
```

Mudar para usuário autenticado

Quando um usuário autenticado faz login em um provedor de identidade e você tem um token, é possível alternar o usuário de não autenticado para autenticado chamando uma função personalizada que atualiza o objeto de credenciais e adiciona o token `logins`.

```
// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
  creds.params.Logins = creds.params.logins || {};
  creds.params.Logins[providerName] = token;

  // Expire credentials to refresh them on the next request
  creds.expired = true;
}
```

Considerações sobre Node.js

Embora o código Node.js seja JavaScript, o uso do AWS SDK for JavaScript em Node.js pode ser diferente do uso do SDK em scripts de navegador. Alguns métodos de API funcionam em Node.js, mas não em scripts de navegador e vice-versa. E usar com êxito algumas APIs depende da familiaridade com padrões de codificação Node.js comuns, como importar e usar outros módulos Node.js como o módulo `File System (fs)`.

Use módulos Node.js integrados

Node.js oferece um conjunto de módulos integrados que é possível usar sem instalá-los. Para usar esses módulos, crie um objeto com o método `require` para especificar o nome do módulo. Por exemplo, para incluir o módulo HTTP integrado, use o seguinte.

```
import http from 'http';
```

Invoque métodos do módulo como se eles fossem métodos desse objeto. Por exemplo, aqui está o código que lê um arquivo HTML.

```
// include File System module
import fs from "fs";
// Invoke readFile method
fs.readFile('index.html', function(err, data) {
  if (err) {
```

```
    throw err;
  } else {
    // Successful file read
  }
});
```

Para obter uma lista completa de todos os módulos integrados fornecidos por Node.js, consulte Documentação do Node.js no site do Node.js.

Use pacotes npm

Além dos módulos integrados, também é possível incluir e incorporar um código de terceiros de `npm`, o gerenciador de pacotes Node.js. Este é um repositório de pacotes de Node.js de código-aberto e uma interface de linha de comando para instalar esses pacotes. Para obter mais informações sobre `npm` e uma lista de pacotes disponíveis no momento, consulte https://www.npmjs.com. Você também pode aprender sobre pacotes adicionais do Node.js que você pode usar aqui GitHub.

Configurar MaxSockets em Node.js

Em Node.js, defina o número máximo de conexões por origem. Se `maxSockets` estiver definido, o cliente HTTP de baixo nível adicionará solicitações HTTP à fila e as atribuirá a soquetes à medida que forem disponibilizados.

Isso permite definir um limite máximo para o número de solicitações simultâneas para uma determinada origem por vez. Reduzir esse valor pode reduzir o número de erros de tempo limite ou de limitação recebidos. No entanto, isso também pode aumentar o uso da memória porque as solicitações serão enfileiradas até um soquete ser disponibilizado.

O exemplo a seguir mostra como definir `maxSockets` para um cliente do DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import https from "https";
let agent = new https.Agent({
  maxSockets: 25
});

let dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    requestTimeout: 3_000,
    httpsAgent: agent
```

```
  });
});
```

O SDK para JavaScript usa um `maxSockets` valor de 50 se você não fornecer um valor ou um `Agent` objeto. Se você fornecer um `Agent` objeto, seu `maxSockets` valor será usado. Para obter mais informações sobre a configuração `maxSockets` no Node.js, consulte a documentação do Node.js.

A partir da v3.521.0 do AWS SDK for JavaScript, você pode usar a seguinte sintaxe abreviada para configurar. `requestHandler`

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
  requestHandler: {
    requestTimeout: 3_000,
    httpsAgent: { maxSockets: 25 },
  },
});
```

Reutilize conexões com keep-alive em Node.js

O agente Node.js HTTP/HTTPS padrão cria uma nova conexão TCP para cada nova solicitação. Para evitar o custo de estabelecer uma nova conexão, o AWS SDK for JavaScript reutiliza conexões TCP por padrão.

Para operações de curta duração, como consultas do Amazon DynamoDB, a sobrecarga de latência da configuração de uma conexão TCP pode ser maior do que a própria operação. Além disso, como a criptografia do DynamoDB em repouso está integrada AWS KMS, você pode enfrentar latências do banco de dados tendo que restabelecer AWS KMS novas entradas de cache para cada operação.

Se você não quiser reutilizar conexões TCP, você pode desativar a reutilização dessas conexões ativas com `keepAlive` cada cliente de serviço, conforme mostrado no exemplo a seguir para um cliente do DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";

const dynamodbClient = new DynamoDBClient({
```

```
    requestHandler: new NodeHttpHandler({
        httpsAgent: new Agent({ keepAlive: false })
    })
});
```

Se `keepAlive` estiver habilitado, você também poderá definir o atraso inicial para pacotes TCP keep-alive com `keepAliveMsecs` que, por padrão, é 1000 ms. Consulte a documentação do Node.js para obter detalhes.

Configurar proxies para Node.js

Se você não conseguir se conectar diretamente à Internet, o SDK para JavaScript suporta o uso de proxies HTTP ou HTTPS por meio de um agente HTTP terceirizado.

Para encontrar um agente HTTP de terceiros, pesquise por “proxy HTTP” em npm.

Para instalar um proxy de agente HTTP de terceiros, digite o seguinte no prompt de comando, em que *`PROXY`* é o nome do pacote de `npm`.

```
npm install PROXY --save
```

Para usar um proxy em seu aplicativo, use as propriedades `httpAgent` e `httpsAgent`, conforme mostrado no exemplo a seguir para um cliente do DynamoDB.

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { HttpsProxyAgent } from "hpagent";
const agent = new HttpsProxyAgent({ proxy: "http://internal.proxy.com" });
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: agent,
        httpsAgent: agent
    }),
});
```

> **Note**
>
> `httpAgent` não é o mesmo que`httpsAgent`, e como a maioria das chamadas do cliente será para `https`, ambas devem ser definidas.

Registre pacotes de certificados em Node.js

Os armazenamentos de confiança padrão de Node.js incluem os certificados necessários para acessar os serviços da AWS . Em alguns casos, pode ser preferível incluir apenas um conjunto específico de certificados.

Neste exemplo, um certificado específico em disco é usado para criar um `https.Agent` que rejeita conexões, a menos que o certificado designado seja fornecido. O recém-criado `https.Agent` é então usado pelo cliente do DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";
import { readFileSync } from "fs";
const certs = [readFileSync("/path/to/cert.pem")];
const agent = new Agent({
  rejectUnauthorized: true,
  ca: certs
});
const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  })
});
```

Considerações sobre o script de navegador

Os tópicos a seguir descrevem considerações especiais sobre o uso dos scripts AWS SDK for JavaScript no navegador.

Tópicos

- Crie o SDK para navegadores
- Compartilhamento de recursos de origem cruzada (CORS)
- Agrupe aplicativos com o webpack

Crie o SDK para navegadores

Ao contrário do SDK para a JavaScript versão 2 (V2), a V3 não é fornecida como um JavaScript arquivo com suporte incluído para um conjunto padrão de serviços. Em vez disso, a V3 permite agrupar e incluir no navegador somente o SDK dos JavaScript arquivos necessários, reduzindo a sobrecarga. Recomendamos usar o Webpack para agrupar o SDK necessário para JavaScript arquivos e quaisquer pacotes adicionais de terceiros necessários em um único `Javascript` arquivo e carregá-lo nos scripts do navegador usando uma tag. `<script>` Para obter mais informações sobre o Webpack, consulte Agrupe aplicativos com o webpack. Para obter um exemplo que usa o Webpack para carregar o SDK V3 JavaScript em um navegador, consulte. Criar um aplicativo para enviar dados para o DynamoDB

Se você trabalha com o SDK fora de um ambiente que aplica o CORS em seu navegador e deseja acessar todos os serviços fornecidos pelo SDK JavaScript, você pode criar uma cópia personalizada do SDK localmente clonando o repositório e executando as mesmas ferramentas de compilação que criam a versão hospedada padrão do SDK. As seções a seguir descrevem as etapas para compilar o SDK com serviços extras e versões da API.

Use o SDK Builder para criar o SDK para JavaScript

> **Note**
>
> O Amazon Web Services versão 3 (V3) não é mais compatível com o Browser Builder. Para minimizar o uso da largura de banda dos aplicativos do navegador, recomendamos que você importe módulos nomeados e os empacote para reduzir o tamanho. Para obter mais informações sobre empacotamento, consulte Agrupe aplicativos com o webpack.

Compartilhamento de recursos de origem cruzada (CORS)

O compartilhamento de recursos de origem cruzada, ou CORS, é um recurso de segurança de navegadores da web modernos. Isso permite que navegadores da web negociem quais domínios podem fazer solicitações de sites ou serviços externos.

CORS é uma consideração importante durante o desenvolvimento de aplicativos de navegador com o AWS SDK for JavaScript porque a maioria das solicitações de recursos é enviada para um domínio externo, como o endpoint de um serviço da web. Se seu JavaScript ambiente impõe a segurança do CORS, você deve configurar o CORS com o serviço.

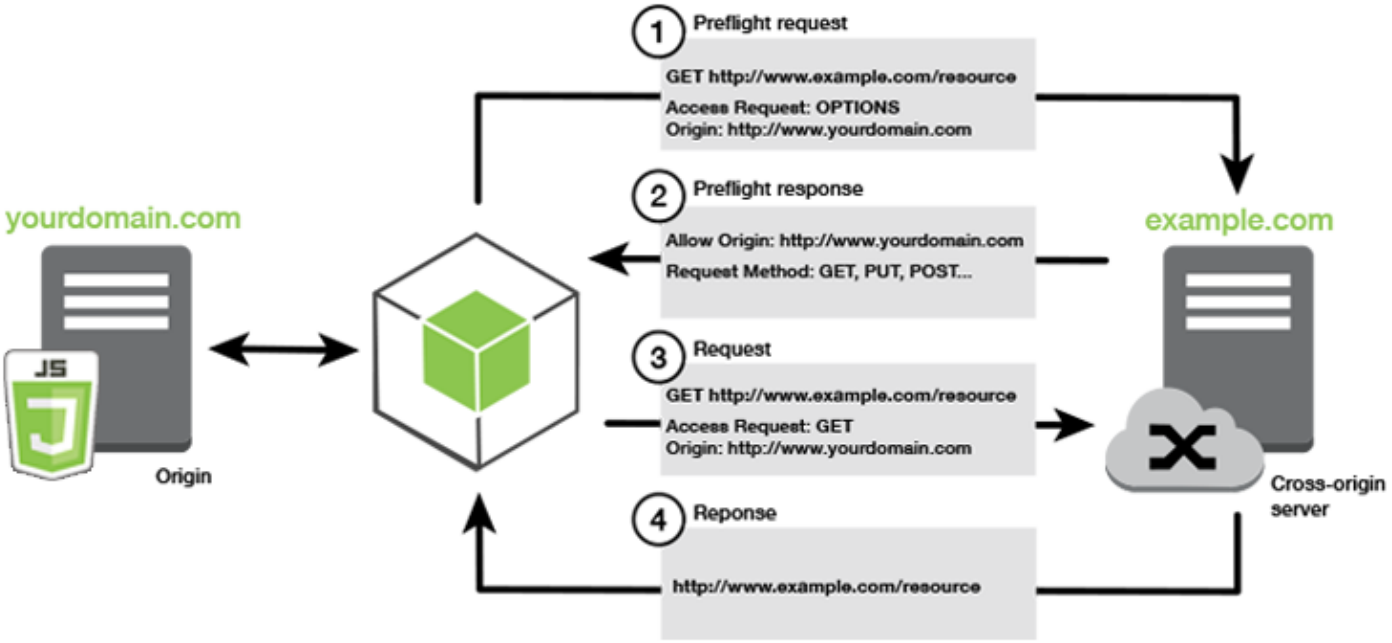
O CORS determina se é necessário permitir o compartilhamento de recursos em uma solicitação entre origens com base em:

- O domínio específico que faz a solicitação
- O tipo de solicitação HTTP feita (GET, PUT, POST, DELETE etc.)

Como funciona o CORS

No caso mais simples, o script de navegador faz uma solicitação GET para um recurso de um servidor em outro domínio. Dependendo da configuração CORS desse servidor, se a solicitação for de um domínio autorizado para enviar solicitações GET, o servidor de origem cruzada responderá retornando o recurso solicitado.

Se o domínio solicitante ou o tipo de solicitação HTTP não estiver autorizado, a solicitação será negada. No entanto, CORS possibilita simular a solicitação antes de enviá-la efetivamente. Neste caso, uma solicitação de simulação é feita em que a operação de solicitação de acesso `OPTIONS` é enviada. Se o servidor de origem cruzada da configuração CORS conceder acesso ao domínio solicitante, o servidor reenviará uma resposta de simulação que lista todos os tipos de solicitação HTTP que o domínio solicitante pode fazer no recurso solicitado.



A configuração do CORS é necessária?

Os buckets do Amazon S3 exigem a configuração de CORS para realizar operações neles. Em alguns JavaScript ambientes, o CORS pode não ser aplicado e, portanto, a configuração do CORS é desnecessária. Por exemplo, se você hospedar o aplicativo de um bucket do e acessar recursos de `*.s3.amazonaws.com` ou algum outro endpoint específico, as solicitações não acessarão um domínio externo. Por isso, essa configuração não exige CORS. Nesse caso, CORS continua sendo usado em serviços que não sejam o Amazon S3.

Configurar o CORS para um bucket do Amazon S3

Você pode configurar um bucket do Amazon S3 para usar o CORS no console do Amazon S3.

Se você estiver configurando o CORS no AWS Web Services Management Console, deverá usar o JSON para criar uma configuração CORS. O novo console de gerenciamento de serviços AWS Web só oferece suporte a configurações JSON CORS.

> **Important**
>
> No novo console de gerenciamento de serviços AWS Web, a configuração do CORS deve ser JSON.

1. No AWS Web Services Management Console, abra o console do Amazon S3, encontre o bucket que você deseja configurar e marque sua caixa de seleção.
2. No painel que é aberto, escolha Permissões.
3. Na guia Permissão, escolha Configuração de CORS.
4. Digite a configuração de CORS no Editor de configuração de CORS e escolha Salvar.

A configuração do CORS é um arquivo XML que contém uma série de regras dentro de um `<CORSRule>`. Uma configuração pode ter até 100 regras. Uma regra é definida por uma das seguintes tags:

- `<AllowedOrigin>`: especifica as origens de domínio permitidas para fazer solicitações entre domínios.
- `<AllowedMethod>`: especifica um tipo de solicitação permitida (GET, PUT, POST, DELETE, HEAD) em solicitações entre domínios.

- `<AllowedHeader>`: especifica os cabeçalhos permitidos em uma solicitação de comprovação.

Para exemplos de configuração, consulte Como configurar CORS no meu bucket? no Guia do usuário do Amazon Simple Storage Service.

Exemplo de configuração do CORS

O exemplo de configuração do CORS a seguir permite que um usuário visualize, adicione, remova ou atualize objetos dentro de um bucket do domínio `example.org`. No entanto, recomendamos definir como escopo `<AllowedOrigin>` para o domínio do seu site. Especifique `"*"` para permitir qualquer origem.

> **Important**
>
> No novo console do S3, a configuração CORS deve ser JSON.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>https://example.org</AllowedOrigin>
    <AllowedMethod>HEAD</AllowedMethod>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
    <ExposeHeader>ETag</ExposeHeader>
    <ExposeHeader>x-amz-meta-custom-header</ExposeHeader>
  </CORSRule>
</CORSConfiguration>
```

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
```

```
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "https://www.example.org"
        ],
        "ExposeHeaders": [
             "ETag",
             "x-amz-meta-custom-header"]
    }
]
```

Essa configuração não autoriza o usuário para executar ações no bucket. Ele permite que o modelo de segurança do navegador faça uma solicitação ao Amazon S3. As permissões devem ser configuradas por meio de permissões do bucket ou permissões da função do IAM.

Use `ExposeHeader` para permitir que os cabeçalhos de resposta de leitura do SDK sejam retornados do Amazon S3. Por exemplo, para ler o cabeçalho `ETag` de um PUT ou de um upload de várias partes, você precisará incluir a tag `ExposeHeader` na configuração, conforme mostrado no exemplo anterior. O SDK só pode acessar cabeçalhos expostos por meio da configuração do CORS. Se você definir metadados no objeto, os valores serão retornados como cabeçalhos com o prefixo `x-amz-meta-`, como `x-amz-meta-my-custom-header`, e também deverão ser expostos da mesma maneira.

Agrupe aplicativos com o webpack

O uso de módulos de código por aplicativos Web nos scripts do navegador ou no Node.js cria dependências. Esses módulos de código podem ter dependências próprias, o que resulta em uma coleção de módulos interligados que seu aplicativo exige para funcionar. Para gerenciar dependências, você pode usar um empacotador de módulos, como `webpack`.

O empacotador de módulos `webpack` analisa o código do seu aplicativo, procurando por instruções `import` ou `require`, para criar pacotes que contenham todos os ativos de que o seu aplicativo precisa. Isso é feito para que os ativos sejam facilmente apresentados em uma página da web. O SDK para JavaScript pode ser incluído `webpack` como uma das dependências a serem incluídas no pacote de saída.

Para obter mais informações sobre`webpack`, consulte o agrupador de módulos webpack em. GitHub

Instale o webpack

Para instalar o empacotador de módulos `webpack`, primeiro você deve ter o npm, o gerenciador de pacotes do Node.js, instalado. Digite o comando a seguir para instalar a `webpack` CLI e JavaScript o módulo.

```
npm install --save-dev webpack
```

Para usar o módulo `path` para trabalhar com caminhos de arquivos e diretórios, que são instalados automaticamente com o webpack, talvez seja necessário instalar o pacote `path-browserify` do Node.js.

```
npm install --save-dev path-browserify
```

Configurar o webpack

Por padrão, o Webpack procura por um JavaScript arquivo nomeado `webpack.config.js` no diretório raiz do seu projeto. Esse arquivo especifica as opções de configuração. Veja a seguir um exemplo de um arquivo de `webpack.config.js` configuração para a WebPack versão 5.0.0 e posterior.

> **Note**
>
> Os requisitos de configuração do Webpack variam dependendo da versão do Webpack que você instala. Para obter mais informações, consulte a documentação do Webpack.

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
```

```
   // Enable WebPack to use the 'path' package.
   resolve:{
  fallback: { path: require.resolve("path-browserify")}
  }
  /**
  * In Webpack version v2.0.0 and earlier, you must tell
  * webpack how to use "json-loader" to load 'json' files.
  * To do this Enter 'npm --save-dev install json-loader' at the
  * command line to install the "json-loader' package, and include the
  * following entry in your webpack.config.js.
  * module: {
    rules: [{test: /\.json$/, use: use: "json-loader"}]
  }
  **/
};
```

Neste exemplo, `browser.js` é especificado como ponto de entrada. O ponto de entrada é o arquivo que o `webpack` usa para iniciar a pesquisa por módulos importados. O nome do arquivo da saída é especificado como `bundle.js`. Esse arquivo de saída conterá tudo o que JavaScript o aplicativo precisa para ser executado. Se o código especificado no ponto de entrada importar ou exigir outros módulos, como o SDK para JavaScript, esse código será agrupado sem a necessidade de especificá-lo na configuração.

Execute o webpack

Para criar um aplicativo para usar o `webpack`, adicione o seguinte ao objeto `scripts` no seu arquivo `package.json`.

```
"build": "webpack"
```

Veja a seguir um exemplo de arquivo `package.json` que demonstra a adição do `webpack`.

```
{
  "name": "aws-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
```

```
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@aws-sdk/client-iam": "^3.32.0",
    "@aws-sdk/client-s3": "^3.32.0"
  },
  "devDependencies": {
    "webpack": "^5.0.0"
  }
}
```

Para criar seu aplicativo, digite o comando a seguir.

```
npm run build
```

Em seguida, o agrupador de `webpack` módulos gera o JavaScript arquivo que você especificou no diretório raiz do seu projeto.

Use o pacote webpack

Para usar o pacote no script de um navegador, você pode incorporar o pacote usando uma tag `<script>`, conforme mostrado no exemplo a seguir.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Amazon SDK with webpack</title>
    </head>
    <body>
        <div id="list"></div>
        <script src="bundle.js"></script>
    </body>
</html>
```

Pacote para Node.js

Você pode usar o `webpack` para gerar pacotes executados no Node.js especificando `node` como um destino na configuração.

```
target: "node"
```

Isso é útil ao executar um aplicativo do Node.js em um ambiente no qual o espaço em disco é limitado. Aqui está um exemplo de configuração do `webpack.config.js` com o Node.js especificado como o destino de saída.

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Let webpack know to generate a Node.js bundle.
  target: "node",
   // Enable WebPack to use the 'path' package.
   resolve:{
  fallback: { path: require.resolve("path-browserify")}
   /**
   * In Webpack version v2.0.0 and earlier, you must tell
   * webpack how to use "json-loader" to load 'json' files.
   * To do this Enter 'npm --save-dev install json-loader' at the
   * command line to install the "json-loader' package, and include the
   * following entry in your webpack.config.js.
   module: {
    rules: [{test: /\.json$/, use: use: "json-loader"}]
  }
  **/
};
```

Trabalhe com AWS serviços no SDK para JavaScript

A AWS SDK for JavaScript v3 fornece acesso aos serviços aos quais oferece suporte por meio de uma coleção de classes de clientes. Com base nessas classes de clientes, você cria objetos de interface de serviço, comumente chamados de objetos de serviço. Cada AWS serviço compatível tem uma ou mais classes de clientes que oferecem APIs de baixo nível para o uso de recursos e recursos do serviço. Por exemplo: as APIs do Amazon DynamoDB estão disponíveis por meio da classe `DynamoDB`.

Os serviços expostos por meio do SDK JavaScript seguem o padrão de solicitação-resposta para trocar mensagens com aplicativos de chamada. Neste padrão, o código que invoca um serviço envia uma solicitação HTTP/HTTPS a um endpoint para o serviço. A solicitação contém os parâmetros necessários para invocar com sucesso o recurso específico que está sendo chamado. O serviço que é invocado gera uma resposta, que é enviada de volta ao solicitante. A resposta contém dados, caso a operação tenha tido sucesso, ou informações de erro, caso a operação não tenha tido sucesso.

A invocação AWS de um serviço inclui todo o ciclo de vida de solicitação e resposta de uma operação em um objeto de serviço, incluindo qualquer tentativa de nova tentativa. Uma solicitação contém zero ou mais propriedades como parâmetros JSON. A resposta é encapsulada em um objeto relacionado à operação e é retornada ao solicitante por meio de uma das várias técnicas, como uma função de retorno de chamada ou uma promessa. JavaScript

Tópicos

Crie e chame objetos de serviço

A JavaScript API é compatível com a maioria dos AWS serviços disponíveis. Cada serviço na JavaScript API fornece a uma classe de cliente um `send` método que você usa para invocar todas

as APIs suportadas pelo serviço. Para obter mais informações sobre classes de serviço, operações e parâmetros na JavaScript API, consulte a Referência da API.

Ao usar o SDK no Node.js, você adiciona o pacote do SDK para cada serviço de que precisa para seu aplicativo usando `import`, que fornece suporte a todos os serviços atuais. O exemplo a seguir cria um objeto de recurso do Amazon S3 na Região `us-west-1`.

```
// Import the Amazon S3 service client
import { S3Client } from "@aws-sdk/client-s3";
// Create an S3 client in the us-west-1 Region
const s3Client = new S3Client({
    region: "us-west-1"
  });
```

Especifique os parâmetros do objeto de serviço

Ao chamar um método de um objeto de serviço, passe os parâmetros em JSON, conforme exigido pela API. Por exemplo, no Amazon S3, para obter um objeto para um bucket e uma chave especificados, passe os seguintes parâmetros para o `GetObjectCommand` método do. `S3Client` Para obter mais informações sobre como passar os parâmetros JSON, consulte Trabalhe com JSON.

```
s3Client.send(new GetObjectCommand({Bucket: 'bucketName', Key: 'keyName'}));
```

Para obter mais informações sobre os parâmetros do Amazon S3, consulte @aws -sdk/client-s3 na Referência da API.

Ligue para os serviços de forma assíncrona

Todas as solicitações feitas por meio do SDK são assíncronas. É importante ter isso em mente ao escrever scripts de navegador. JavaScript a execução em um navegador da Web normalmente tem apenas um único thread de execução. Depois de fazer uma chamada assíncrona para um AWS serviço, o script do navegador continua em execução e, no processo, pode tentar executar o código que depende desse resultado assíncrono antes que ele retorne.

Fazer chamadas assíncronas para um AWS serviço inclui gerenciar essas chamadas para que seu código não tente usar dados antes que eles estejam disponíveis. Os tópicos desta seção explicam a necessidade de gerenciar chamadas assíncronas e detalha diferentes técnicas que você pode usar para gerenciá-las.

Embora você possa usar qualquer uma dessas técnicas para gerenciar chamadas assíncronas, recomendamos usar async/await para todos os novos códigos.

async/await

Recomendamos que você use essa técnica, pois é o comportamento padrão na V3.

promise

Use essa técnica em navegadores que não são compatíveis com async/await.

callback

Evite usar callbacks, exceto em casos muito simples. No entanto, você pode achar que é útil para cenários de migração.

Tópicos

Gerencie chamadas assíncronas

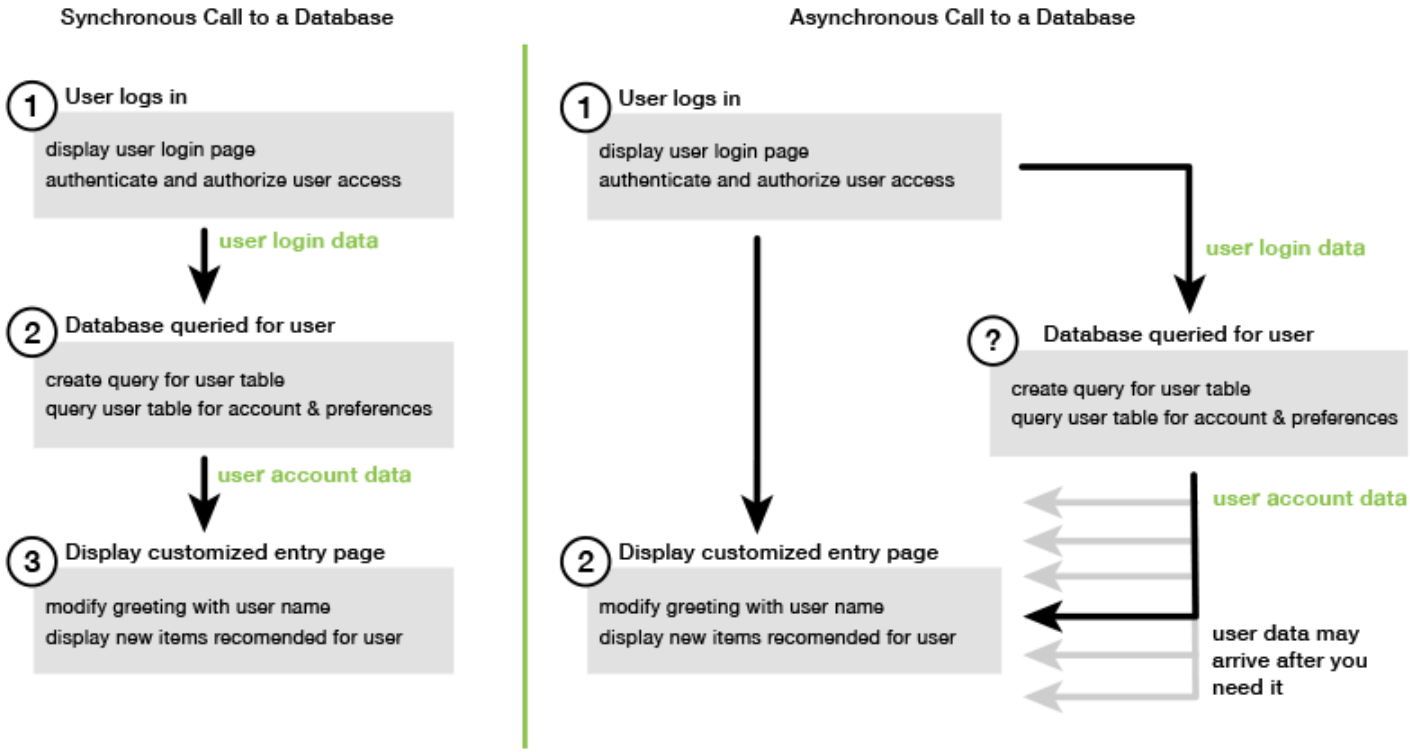
Por exemplo, a página inicial de um site de comércio eletrônico permite que os clientes que retornam façam login. Parte do benefício para os clientes que fazem login é que, depois de fazerem-no, o site se personaliza de acordo com suas preferências específicas. Para fazer isso acontecer:

1. O cliente deve fazer login e ser validado com suas credenciais de login.
2. As preferências do cliente são solicitadas a banco de dados de clientes.
3. O banco de dados fornece as preferências do cliente, que são usadas para personalizar o site antes que a página carregue.

Se essas tarefas forem executadas de forma síncrona, cada uma delas deverá terminar antes de a seguinte começar. A página da Web não terminaria de carregar até que as preferências do cliente fossem apresentadas pelo banco de dados. No entanto, após a consulta de banco de dados ser enviada ao servidor, o recebimento dos dados do cliente pode ser atrasado ou até mesmo falhar

devido a gargalos da rede, tráfego excepcionalmente alto no banco de dados ou conexão ruim nos dispositivos móveis.

Para evitar que o site congele sob essas condições, chame o banco de dados de forma assíncrona. Depois de a chamada do banco de dados ser executada, enviando sua solicitação assíncrona, o código continuará a ser executado conforme o esperado. Se você não gerir adequadamente a resposta de uma chamada assíncrona, o código poderá tentar usar informações que espera de volta do banco de dados quando esses dados ainda não estiverem disponíveis.



Use async/await

Em vez de usar promessas, considere o uso de async/await. As funções assíncronas são mais simples e usam menos boilerplate do que as promessas. Await só pode ser usado em uma função assíncrona para esperar assincronamente um valor.

O exemplo a seguir usa async/await para listar todas as tabelas do Amazon DynamoDB em `us-west-2`.

> **Note**
>
> Para executar este exemplo:

- Instale o AWS SDK for JavaScript cliente do DynamoDB `npm install @aws-sdk/client-dynamodb` entrando na linha de comando do seu projeto.
- Verifique se você configurou suas AWS credenciais corretamente. Para ter mais informações, consulte Definir credenciais.

```
import { DynamoDBClient,
ListTablesCommand } from "@aws-sdk/client-dynamodb";
(async function () {
    const dbClient = new DynamoDBClient({ region: "us-west-2" });
    const command = new ListTablesCommand({});

    try {
        const results = await dbClient.send(command);
        console.log(results.TableNames.join('\n'));
    } catch (err) {
        console.error(err)
    }
})();
```

> **Note**
>
> Nem todos os navegadores oferecem suporte para async/await. Consulte Funções assíncronas para obter uma lista de navegadores com suporte para async/await.

Use JavaScript promessas

Use o método AWS SDK for JavaScript v3 (`ListTablesCommand`) do cliente de serviço para fazer a chamada de serviço e gerenciar o fluxo assíncrono em vez de usar retornos de chamada. O exemplo a seguir mostra como obter os nomes de tabelas do Amazon DynamoDB em `us-west-2`.

```
import { DynamoDBClient,
        ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbClient = new DynamoDBClient({ region: 'us-west-2' });

dbClient
  .listtables(new ListTablesCommand({}))
```

```
  .then(response => {
    console.log(response.TableNames.join('\n'));
  })
  .catch((error) => {
    console.error(error);
  });
```

Coordene várias promessas

Em algumas situações, seu código deve fazer várias chamadas assíncronas que exigem ação somente quando todos tiverem sido retornados com êxito. Se você gerenciar as chamadas individuais do método assíncrono com promessas, pode criar uma promessa adicional que usa o método `all`.

Esse método cumpre essa promessa generalista se, e quando, a série de promessas que você passar para o método forem cumpridas. A função de retorno de chamada é transmitida a um array dos valores das promessas passadas para o método `all`.

No exemplo a seguir, uma AWS Lambda função deve fazer três chamadas assíncronas para o Amazon DynamoDB, mas só pode ser concluída depois que as promessas de cada chamada forem cumpridas.

```
const values = await Promise.all([firstPromise, secondPromise, thirdPromise]);

console.log("Value 0 is " + values[0].toString);
console.log("Value 1 is " + values[1].toString);
console.log("Value 2 is " + values[2].toString);

return values;
```

Suporte de navegador e Node.js para promessas

Support para JavaScript promessas nativas (ECMAScript 2015) depende do JavaScript mecanismo e da versão em que seu código é executado. Para ajudar a determinar o suporte para JavaScript promessas em cada ambiente em que seu código precisa ser executado, consulte a tabela de compatibilidade do ECMAScript em. GitHub

Use uma função de retorno de chamada anônima

Cada método de objeto de serviço pode aceitar uma função de retorno de chamada anônima como o último parâmetro. A assinatura dessa função de retorno de chamada é:

```
function(error, data) {
    // callback handling code
};
```

Essa função de retorno de chamada é executada ao se retornar uma resposta bem-sucedida ou dados de erro. Se a chamada do método for bem-sucedida, o conteúdo da resposta estará disponível para a função de retorno de chamada no parâmetro `data`. Se a chamada não for bem-sucedida, os detalhes sobre a falha são fornecidos no parâmetro `error`.

Normalmente o código dentro da função de retorno de chamada testa um erro, que ele processa, caso seja retornado um erro. Se o erro não for retornado, o código recuperará os dados na resposta pelo parâmetro `data`. O formato básico da função de retorno de chamada é semelhante a este exemplo.

```
function(error, data) {
    if (error) {
        // error handling code
        console.log(error);
    } else {
        // data handling code
        console.log(data);
    }
};
```

No exemplo anterior, os detalhes do erro ou dos dados retornados são registrados no console. Veja a seguir um exemplo que mostra uma função de retorno de chamada passada como parte da chamada de um método em um objeto de serviço.

```
ec2.describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
  } else {
    console.log(data); // request succeeded
  }
});
```

Crie solicitações de serviço para clientes

Fazer solicitações aos clientes AWS de atendimento é simples. A versão 3 (V3) do SDK JavaScript permite que você envie solicitações.

> **Note**
>
> Você também pode realizar operações usando os comandos da versão 2 (V2) ao usar a V3 do SDK para. JavaScript Para ter mais informações, consulte Usando comandos v2.

Para enviar uma solicitação:

1. Inicialize um objeto cliente com a configuração desejada, como uma Região da AWS específica.
2. (Opcional) Crie um objeto JSON de solicitação com os valores da solicitação, como o nome de um bucket específico do Amazon S3. Você pode examinar os parâmetros da solicitação consultando o tópico Referência da API referente à interface com o nome associado ao método do cliente. Por exemplo, se você usar o método *`AbcCommand`*client, a interface de solicitação será *`AbcInput`*.
3. Inicialize um comando de serviço, opcionalmente, com o objeto de solicitação como entrada.
4. Chame `send` no cliente com o objeto de comando como entrada.

Por exemplo, para listar suas tabelas do Amazon DynamoDB em `us-west-2`, você pode fazer isso com async/await.

```
  import {
  DynamoDBClient,
  ListTablesCommand
  } from "@aws-sdk/client-dynamodb";

  (async function() {
  const dbClient = new DynamoDBClient({ region: 'us-west-2' });
  const command = new ListTablesCommand({});

  try {
    const results = await dbClient.send(command);
    console.log(results.TableNames.join('\n'));
  } catch (err) {
    console.error(err);
  }
})();
```

Gerencie as respostas do cliente de serviço

Depois que um método de cliente de serviço foi chamado, ele retorna uma instância do objeto de resposta de uma interface com o nome associado ao método de cliente. Por exemplo, se você usar o método *AbcCommand*client, o objeto de resposta será do tipo *AbcResponse*(interface).

Dados de acesso retornados na resposta

O objeto de resposta contém os dados, como propriedades, retornados pela solicitação de serviço.

Em Crie solicitações de serviço para clientes, o comando `ListTablesCommand` retornou os nomes de tabela na propriedade `TableNames` da resposta.

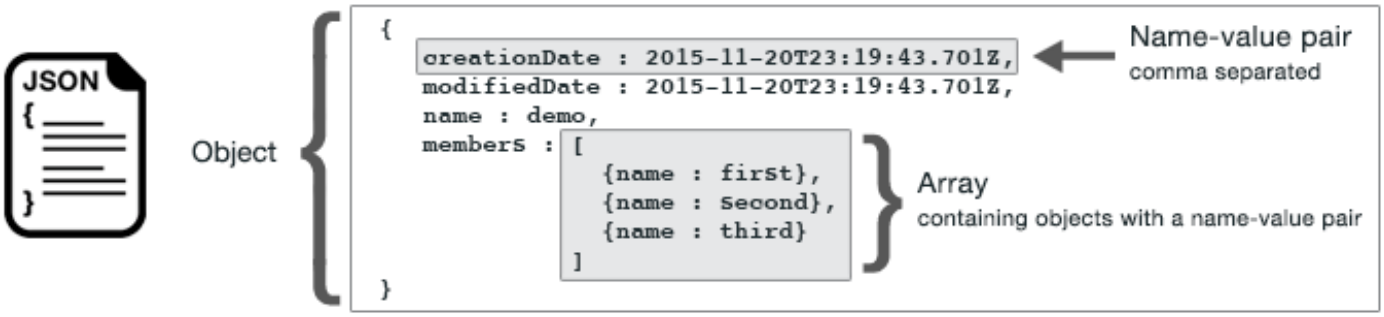
Informações de erro de acesso

Se um comando falhar, causará uma exceção. Você pode lidar com a exceção conforme necessário.

Trabalhe com JSON

JSON é um formato de intercâmbio de dados capaz de ser lido por humanos e por máquina. Embora o nome JSON seja um acrônimo para JavaScript Object Notation, o formato do JSON é independente de qualquer linguagem de programação.

O AWS SDK for JavaScript usa JSON para enviar dados para objetos de serviço ao fazer solicitações e recebe dados de objetos de serviço como JSON. Para mais informações sobre JSON, consulte json.org.



JSON representa dados de duas formas:

- Um objeto, que é uma coleção não ordenada de pares de nome/valor. Um objeto é definido dentro das chaves esquerda ({) e direita (}). Cada par de nome e valor começa com o nome seguido por uma vírgula seguido pelo valor. O pares de nome/valor são separados por vírgulas.

- Um array, que é uma coleção ordenada de valores. Um array é definido dentro dos colchetes esquerdo (`[`) e direito (`]`). Os itens no array são separados por vírgulas.

Aqui está um exemplo de um objeto JSON que contém um array de objetos em que os objetos representam as cartas de um baralho. Cada carta é definida por dois pares de nome/valor: um que especifica um valor exclusivo para identificar que essa carta e outro que especifica um URL que aponta para a imagem da carta correspondente.

```
var cards = [
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"}
];
```

JSON como parâmetros do objeto de serviço

Veja a seguir um exemplo de JSON simples usado para definir os parâmetros de uma chamada para um objeto de serviço do AWS Lambda .

```
const params = {
   FunctionName : funcName,
   Payload : JSON.stringify(payload),
   LogType : LogType.Tail,
};
```

O objeto `params` é definido por três pares de nome/valor, separados por vírgulas, dentro das chaves esquerda e direita. Ao fornecer parâmetros para uma chamada do método do objeto de serviço, os nomes são determinados pelos nomes de parâmetro do método do objeto de serviço que você pretende chamar. Ao invocar uma função do Lambda, `FunctionName`, `Payload` e `LogType` são os parâmetros usados para chamar o método `invoke` em um objeto de serviço do Lambda.

Ao passar os parâmetros para uma chamada do método do objeto de serviço, forneça o objeto JSON para a chamada de método, conforme mostrado no exemplo a seguir de como invocar uma função do Lambda.

```
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 };
};
```

SDK para exemplos de JavaScript código

Os tópicos desta seção contêm exemplos de como usar o AWS SDK for JavaScript com as APIs de vários serviços para realizar tarefas comuns.

Encontre o código-fonte desses e de outros exemplos no Repositório de exemplos de AWS código em GitHub. Para propor um novo exemplo de código para a equipe de AWS documentação considerar a produção, crie uma solicitação. A equipe está buscando produzir exemplos de código que abrangem cenários e casos de uso mais amplos, em vez de trechos de código simples que abrangem apenas chamadas de API individuais. Para obter instruções, consulte a seção Código de autoria nas diretrizes de contribuição em GitHub.

> **Important**
>
> Esses exemplos usam a sintaxe import/export de ECMAScript6.
>
> - Isso requer o Node.js versão 14.17 ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript para obter as diretrizes de conversão.

Tópicos

- Sintaxe ES6/CommonJS de JavaScript
- Exemplos do Amazon DynamoDB
- Exemplos do AWS Elemental MediaConvert

- Exemplos do AWS Lambda
- Exemplos do Amazon Lex
- Exemplos do Amazon Polly
- Exemplos do Amazon Redshift
- Exemplos do Amazon Simple Email Service
- Exemplos do Amazon Simple Notification Service
- Exemplos do Amazon Transcribe
- Configuração do Node.js em uma instância do Amazon EC2
- Criar um aplicativo para enviar dados para o DynamoDB
- Criar um aplicativo de transcrição com usuários autenticados
- Invocar o Lambda com o API Gateway
- Criação de fluxos de trabalho sem servidor da AWS usando o AWS SDK for JavaScript
- Criação de eventos programados para executar funções do AWS Lambda
- Como criar um chatbot do Amazon Lex
- Criação de um exemplo de aplicativo de mensagens

Sintaxe ES6/CommonJS de JavaScript

Os exemplos de código do AWS SDK for JavaScript são escritos em ECMAScript 6 (ES6). O ES6 apresenta nova sintaxe e novos recursos para tornar seu código mais moderno e legível, além de fazer mais.

O ES6 requer que você use o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js. Entretanto, se você preferir, poderá converter qualquer um dos nossos exemplos em sintaxe CommonJS usando as seguintes diretrizes:

- Remova `"type" : "module"` do `package.json` no ambiente do projeto.
- Converta todas as instruções `import` de ES6 em instruções `require` de CommonJS. Por exemplo, converta:

```
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "./libs/s3Client.js";
```

Em seu equivalente de CommonJS:

```
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("./libs/s3Client.js");
```

- Converta todas as instruções `export` de ES6 em instruções `module.exports` de CommonJS. Por exemplo, converta:

```
export {s3}
```

Em seu equivalente de CommonJS:

```
module.exports = {s3}
```

O exemplo a seguir demonstra o exemplo de código para criar um bucket do Amazon S3 em ES6 e CommonJS.

ES6

libs/s3Client.js

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS region
const REGION = "eu-west-1"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
export {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using ES6 syntax.
 import { CreateBucketCommand } from "@aws-sdk/client-s3";
 import { s3 } from "./libs/s3Client.js";
```

```
// Get service clients module and commands using CommonJS syntax.
// const { CreateBucketCommand } = require("@aws-sdk/client-s3");
// const { s3 } = require("./libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

CommonJS

libs/s3Client.js

```
// Create service client module using CommonJS syntax.
 const { S3Client } = require("@aws-sdk/client-s3");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
 // Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
 module.exports ={s3};
```

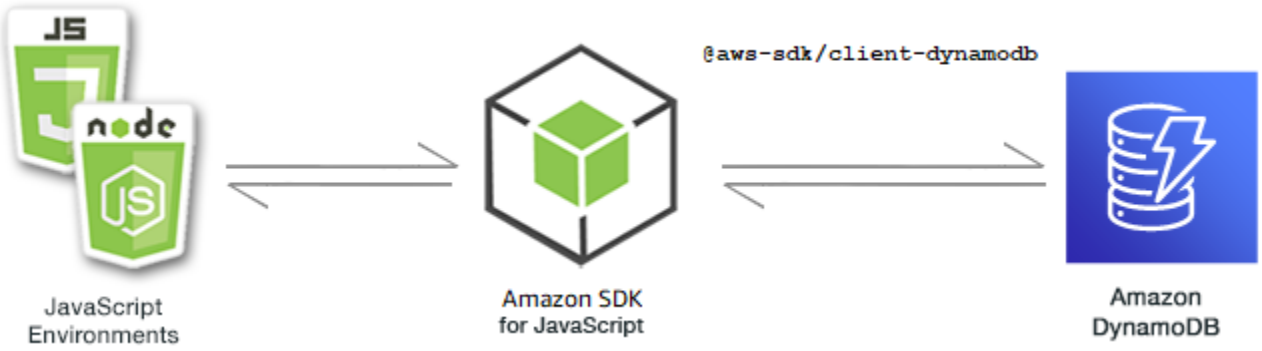
s3_createbucket.js

```
// Get service clients module and commands using CommonJS syntax.
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("./libs/s3Client.js");
```

```
// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Exemplos do Amazon DynamoDB

Amazon DynamoDB é um banco de dados em nuvem NoSQL totalmente gerenciado que oferece suporte a modelos de armazenamento de documentos e chave-valor. Você cria tabelas sem esquema de dados sem a necessidade de provisionar ou manter servidores de banco de dados dedicados.



A API JavaScript do DynamoDB é exposta por meio das classes cliente `DynamoDB`, `DynamoDBStreams` e `DynamoDB.DocumentClient`. Para obter mais informações sobre como usar as classes cliente do DynamoDB, consulte Classe: DynamoDB, Classe: DynamoDBStreams e Classe: DynamoDB utility na Referência da API.

Tópicos

- Criação e uso de tabelas no DynamoDB

- Ler e gravar um único item no DynamoDB
- Ler e gravar itens em lote no DynamoDB
- Consultar e verificar uma tabela do DynamoDB
- Uso do cliente de documentos do DynamoDB

Criação e uso de tabelas no DynamoDB

Este exemplo de código Node.js mostra:

- Como criar e gerenciar tabelas usadas para armazenar e recuperar dados do DynamoDB.

O cenário

De forma semelhante a outros sistemas de banco de dados, o DynamoDB armazena dados em tabelas. Uma tabela do DynamoDB é uma coleção de dados organizada em itens semelhantes às linhas. Para armazenar ou acessar dados no DynamoDB, você cria e trabalha com tabelas.

Neste exemplo, você usa uma série de módulos de Node.js para realizar operações básicas com uma tabela do DynamoDB. O código usa o SDK para JavaScript para criar e trabalhar com tabelas usando esses métodos da classe de cliente `DynamoDB`:

- `CreateTableCommand`
- `ListTablesCommand`
- `DescribeTableCommand`
- `DeleteTableCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node.js e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.

- Instale o cliente DynamoDB do SDK para JavaScript. Para obter mais informações, consulte Novidades da versão 3.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Esses exemplos usam ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js. No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

> **Note**
>
> Para obter informações sobre os tipos de dados usados nesses exemplos, consulte Tipos de dados e regras de nomenclatura compatíveis no Amazon DynamoDB.

Criar uma tabela

Crie um módulo do Node.js com o nome de arquivo `create-table.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para criar uma tabela, que, neste exemplo, inclui o nome e o tipo de dados de cada atributo, o esquema de chave, o nome da tabela e as unidades de taxa de transferência para provisionar. Chame o método `CreateTableCommand` do objeto de serviço do DynamoDB.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
```

```
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node create-table.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Listar as tabelas

Crie um módulo do Node.js com o nome de arquivo `list-tables.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON que contém os parâmetros necessários para listar as tabelas, que, neste exemplo, limita o número de tabelas listadas a 10. Chame o método `ListTablesCommand` do objeto de serviço do DynamoDB.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node list-tables.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Descrever uma tabela

Crie um módulo do Node.js com o nome de arquivo `describe-table.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente DynamoDB. Crie um objeto JSON contendo os parâmetros necessários para descrever um método `DescribeTableCommand` do objeto de serviço do DynamoDB.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node describe-table.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Excluir uma tabela

Crie um módulo do Node.js com o nome de arquivo `delete-table.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON com os parâmetros necessários para excluir uma tabela, que, neste exemplo, inclui o nome da tabela fornecido como um parâmetro de linha de comando. Chame o método `DeleteTableCommand` do objeto de serviço do DynamoDB.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node delete-table.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Ler e gravar um único item no DynamoDB

Este exemplo de código Node.js mostra:

- Como adicionar um item em uma tabela do DynamoDB.
- Como recuperar um item em uma tabela do DynamoDB.
- Como excluir um item de uma tabela do DynamoDB.

O cenário

Neste exemplo, você usa uma série de módulos de Node.js para ler e gravar um item em uma tabela do DynamoDB usando esses métodos da classe de cliente `DynamoDB`:

- PutItemCommand
- UpdateItemCommand
- GetItemCommand
- DeleteItemCommand

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node.js e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.
- Crie uma tabela do DynamoDB cujos itens você possa acessar. Para obter mais informações sobre como criar uma tabela do DynamoDB, consulte Criação e uso de tabelas no DynamoDB.

> **Important**
>
> Esses exemplos usam ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js. No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

> **Note**
>
> Para obter informações sobre os tipos de dados usados nesses exemplos, consulte Tipos de dados e regras de nomenclatura compatíveis no Amazon DynamoDB.

Gravar um item

Crie um módulo do Node.js com o nome de arquivo `put-item.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON que contém os parâmetros necessários para adicionar um item, que, neste exemplo, inclui o nome da tabela e um mapa que define os atributos a serem configurados e os valores de cada atributo. Chame o método `PutItemCommand` do objeto de serviço do cliente DynamoDB.

```
import { PutItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new PutItemCommand({
    TableName: "Cookies",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Item: {
      Flavor: { S: "Chocolate Chip" },
      Variants: { SS: ["White Chocolate Chip", "Chocolate Chunk"] },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node put-item.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Atualizar um item

Crie um módulo do Node.js com o nome de arquivo `update-item.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON que contém os parâmetros necessários para adicionar um item, que, neste exemplo, inclui o nome da tabela, a chave a ser atualizada e a expressão de data que mapeia os novos nomes de atributo e os valores para cada novo atributo. Chame o método `UpdateItemCommand` do objeto de serviço do cliente DynamoDB.

```
import { UpdateItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new UpdateItemCommand({
    TableName: "IceCreams",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Flavor: { S: "Vanilla" },
    },
    UpdateExpression: "set HasChunks = :chunks",
    ExpressionAttributeValues: {
      ":chunks": { BOOL: "false" },
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node update-item.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Obter um item

Crie um módulo do Node.js com o nome de arquivo `get-item.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Para identificar o item a ser obtido, você deve fornecer o valor da chave primária desse item na tabela. Por padrão, o método `GetItemCommand` retorna todos os valores de atributo definidos para o item. Para obter apenas um subconjunto de todos os valores de atributo possíveis, especifique uma expressão de projeção.

Crie um objeto JSON que contenha os parâmetros necessários para obter um item, que, neste exemplo, inclui o nome da tabela, o nome e o valor da chave do item que você está recebendo, e uma expressão de projeção que identifica o atributo do item que deseja recuperar. Chame o método `GetItemCommand` do objeto de serviço do cliente DynamoDB.

O exemplo de código a seguir recupera um item de uma tabela com uma chave primária composta somente por uma chave de partição e não por uma chave de partição e uma de classificação. Se a tabela tiver uma chave primária composta por uma chave de partição e uma chave de classificação, você também deverá especificar o nome e o atributo da chave de classificação.

```
import { GetItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new GetItemCommand({
    TableName: "CafeTreats",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      TreatId: { N: "101" },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
```

```
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node get-item.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Excluir um item

Crie um módulo do Node.js com o nome de arquivo `delete-item.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON com os parâmetros necessários para excluir um item, que, neste exemplo, inclui o nome da tabela, além do nome da chave e o valor do item que você está excluindo. Chame o método `DeleteItemCommand` do objeto de serviço do cliente DynamoDB.

```
import { DeleteItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteItemCommand({
    TableName: "Drinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Name: { S: "Pumpkin Spice Latte" },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node delete-item.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Ler e gravar itens em lote no DynamoDB

Este exemplo de código Node.js mostra:

- Como ler e gravar lotes de itens em uma tabela do DynamoDB.

O cenário

Neste exemplo, você usa uma série de módulos de Node.js para colocar um lote de itens em uma tabela do DynamoDB, bem como ler um lote de itens. O código usa o SDK para JavaScript para realizar operações de leitura e gravação em lote usando estes métodos da classe de cliente DynamoDB:

- BatchGetItemCommand
- BatchWriteItemCommand

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.
- Crie uma tabela do DynamoDB cujos itens você possa acessar. Para obter mais informações sobre como criar uma tabela do DynamoDB, consulte Criação e uso de tabelas no DynamoDB.

> **Important**
>
> Esses exemplos usam ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js. No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

> **Note**
>
> Para obter informações sobre os tipos de dados usados nesses exemplos, consulte Tipos de dados e regras de nomenclatura compatíveis no Amazon DynamoDB.

Ler itens em um lote

Crie um módulo do Node.js com o nome de arquivo `batch-get-item.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para obter um lote de itens, que, neste exemplo, inclui o nome de uma ou mais tabelas para leitura, os valores de chaves para leitura em cada tabela, e a expressão de projeção que especifica os atributos a serem retornados. Chame o método `BatchGetItemCommand` do objeto de serviço do DynamoDB.

```
import { BatchGetItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new BatchGetItemCommand({
    RequestItems: {
      // Each key in this object is the name of a table. This example refers
      // to a PageAnalytics table.
      PageAnalytics: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            // "PageName" is the partition key (simple primary key).
            // "S" specifies a string as the data type for the value "Home".
            // For more information about data types,
```

```
            // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
            // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
            PageName: { S: "Home" },
          },
          {
            PageName: { S: "About" },
          },
        ],
        // Only return the "PageName" and "PageViews" attributes.
        ProjectionExpression: "PageName, PageViews",
      },
    },
  });

  const response = await client.send(command);
  console.log(response.Responses["PageAnalytics"]);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node batch-get-item.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Gravar itens em um lote

Crie um módulo do Node.js com o nome de arquivo `batch-write-item.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para obter um lote de itens, que, neste exemplo, inclui a tabela na qual você deseja gravar itens, as chaves que você deseja gravar para cada item, e os atributos com os valores. Chame o método `BatchWriteItemCommand` do objeto de serviço do DynamoDB.

```
import {
  BatchWriteItemCommand,
  DynamoDBClient,
} from "@aws-sdk/client-dynamodb";
```

```
const client = new DynamoDBClient({});

export const main = async () => {
  const command = new BatchWriteItemCommand({
    RequestItems: {
      // Each key in this object is the name of a table. This example refers
      // to a Coffees table.
      Coffees: [
        // Each entry in Coffees is an object that defines either a PutRequest or
 DeleteRequest.
        {
          // Each PutRequest object defines one item to be inserted into the table.
          PutRequest: {
            // The keys of Item are attribute names. Each attribute value is an object
 with a data type and value.
            // For more information about data types,
            // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes
            Item: {
              Name: { S: "Donkey Kick" },
              Process: { S: "Wet-Hulled" },
              Flavors: { SS: ["Earth", "Syrup", "Spice"] },
            },
          },
        },
        {
          PutRequest: {
            Item: {
              Name: { S: "Flora Ethiopia" },
              Process: { S: "Washed" },
              Flavors: { SS: ["Stone Fruit", "Toasted Almond", "Delicate"] },
            },
          },
        },
      ],
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node batch-write-item.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Consultar e verificar uma tabela do DynamoDB

Este exemplo de código Node.js mostra:

- Como consultar e verificar uma tabela do DynamoDB em busca de itens.

O cenário

Consultar encontra itens em uma tabela ou um índice secundário usando apenas valores de atributo de chave primária. Você deve fornecer um nome da chave de partição e um valor a serem procurados. Você pode fornecer um nome da chave e um valor de classificação, além de usar um operador de comparação para refinar os resultados da pesquisa. Pesquisar encontra itens ao verificar todos os itens na tabela especificada.

Neste exemplo, você usa uma série de módulos Node.js para identificar um ou mais itens que queira recuperar de uma tabela do DynamoDB. O código usa o SDK para JavaScript para consultar e verificar tabelas usando estes métodos da classe de cliente do DynamoDB:

- QueryCommand
- ScanCommand

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node.js e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.
- Crie uma tabela do DynamoDB cujos itens você possa acessar. Para obter mais informações sobre como criar uma tabela do DynamoDB, consulte Criação e uso de tabelas no DynamoDB.

Important

Esses exemplos usam ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js. No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Note

Para obter informações sobre os tipos de dados usados nesses exemplos, consulte Tipos de dados e regras de nomenclatura compatíveis no Amazon DynamoDB.

Consultar uma tabela

Crie um módulo do Node.js com o nome de arquivo `query.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para consultar a tabela, que, neste exemplo, inclui o nome da tabela, o `ExpressionAttributeValues` necessário pela consulta, um `KeyConditionExpression` que usa esses valores para definir quais itens a consulta retorna, e os nomes dos valores de atributo a serem retornados para cada item. Chame o método `QueryCommand` do objeto de serviço do DynamoDB.

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new QueryCommand({
    KeyConditionExpression: "Flavor = :flavor",
    // For more information about data types,
```

```
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":flavor": { S: "Key Lime" },
      ":searchKey": { S: "no coloring" },
    },
    FilterExpression: "contains (Description, :searchKey)",
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node query.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Verificar uma tabela

Crie um módulo do Node.js com o nome de arquivo `scan.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Para acessar o DynamoDB, crie um objeto de serviço do cliente `DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para verificar a tabela em busca de itens, que, neste exemplo, inclui o nome da tabela, a lista de valores de atributos a serem retornados para cada item correspondente e uma expressão para filtrar o conjunto de resultados a fim de encontrar itens que contenham uma frase especificada. Chame o método `ScanCommand` do objeto de serviço do DynamoDB.

```
import { DynamoDBClient, ScanCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});
```

```
export const main = async () => {
  const command = new ScanCommand({
    FilterExpression: "CrustType = :crustType",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":crustType": { S: "Graham Cracker" },
    },
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node scan.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Uso do cliente de documentos do DynamoDB

Este exemplo de código Node.js mostra:

- Como acessar uma tabela do DynamoDB usando os utilitários do DynamoDB.

O cenário

O cliente de documento do DynamoDB simplifica o trabalho com itens abstraindo a noção de valores de atributo. Essa abstração anota tipos JavaScript nativos fornecidos como parâmetros de entrada, bem como converte dados de resposta anotados em tipos JavaScript nativos.

Para obter mais informações sobre o cliente de documento do DynamoDB, consulte @aws-sdk/lib-dynamodb README no GitHub. Para obter mais informações sobre programação com o Amazon DynamoDB, consulte Programação com o DynamoDB no Guia do desenvovledor do Amazon DynamoDB.

Neste exemplo, você usa uma série de módulos de Node.js para realizar operações básicas em uma tabela do DynamoDB usando utilitários do DynamoDB. O código usa o SDK para JavaScript para consultar e verificar tabelas usando estes métodos da classe de cliente de documento do DynamoDB:

- GetCommand
- PutCommand
- UpdateCommand
- QueryCommand
- DeleteCommand

Para obter mais informações sobre como configurar o cliente de documento do DynamoDB, consulte @aws-sdk/lib-dynamodb.

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node.js e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.
- Crie uma tabela do DynamoDB cujos itens você possa acessar. Para obter mais informações sobre como criar uma tabela do DynamoDB usando o SDK para JavaScript, consulte Criação e uso de tabelas no DynamoDB. Também é possível usar o console do DynamoDB para criar uma tabela.

Important

Esses exemplos usam ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js. No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Note

Para obter informações sobre os tipos de dados usados nesses exemplos, consulte Tipos de dados e regras de nomenclatura compatíveis no Amazon DynamoDB.

Obtenção de um item de uma tabela

Crie um módulo do Node.js com o nome de arquivo `get.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Isso inclui `@aws-sdk/lib-dynamodb`, um pacote de biblioteca que fornece funcionalidade de cliente de documentos para `@aws-sdk/client-dynamodb`. Em seguida, defina a configuração conforme mostrado abaixo para realizar marshalling e unmarshalling - como um segundo parâmetro opcional - durante a criação do cliente do documento. Em seguida, crie os clientes. Agora, crie um objeto JSON que contenha os parâmetros necessários para obter um item da tabela, que, neste exemplo, inclui o nome da tabela, o nome da chave hash nessa tabela e o valor da chave hash do item que você deseja receber. Chame o método `GetCommand` do cliente de documento do DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node get.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Colocar um item em uma tabela

Crie um módulo do Node.js com o nome de arquivo `put.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Isso inclui `@aws-sdk/lib-dynamodb`, um pacote de biblioteca que fornece funcionalidade de cliente de documentos para `@aws-sdk/client-dynamodb`. Em seguida, defina a configuração conforme mostrado abaixo para realizar marshalling e unmarshalling - como um segundo parâmetro opcional - durante a criação do cliente do documento. Em seguida, crie os clientes. Crie um objeto JSON que contenha os parâmetros necessários para gravar um item na tabela, que, neste exemplo, inclui o nome da tabela e uma descrição do item a ser adicionado ou atualizado que inclua a chave hash e o valor, bem como nomes e valores de atributos a serem definidos no item. Chame o método `PutCommand` do cliente de documento do DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
```

```
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node put.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Atualizar um item em uma tabela

Crie um módulo do Node.js com o nome de arquivo `update.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Isso inclui `@aws-sdk/lib-dynamodb`, um pacote de biblioteca que fornece funcionalidade de cliente de documentos para `@aws-sdk/client-dynamodb`. Em seguida, defina a configuração conforme mostrado abaixo para realizar marshalling e unmarshalling - como um segundo parâmetro opcional - durante a criação do cliente do documento. Em seguida, crie os clientes. Crie um objeto JSON que contenha os parâmetros necessários para gravar um item na tabela, que, neste exemplo, inclui o nome da tabela, a chave do item a ser atualizada, um conjunto de `UpdateExpressions` que definem os atributos do item a ser atualizado com tokens com valores atribuídos nos parâmetros `ExpressionAttributeValues`. Chame o método `UpdateCommand` do cliente de documento do DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node update.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Consultar uma tabela

Crie um módulo do Node.js com o nome de arquivo `query.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Isso inclui `@aws-sdk/lib-dynamodb`, um pacote de biblioteca que fornece funcionalidade de cliente de documentos para `@aws-sdk/client-dynamodb`. Crie um objeto JSON que contenha os parâmetros necessários para consultar a tabela, que, neste exemplo, inclui o nome da tabela, o `ExpressionAttributeValues` necessário pela consulta e um `KeyConditionExpression` que usa esses valores para definir quais itens a consulta retorna. Chame o método `QueryCommand` do cliente de documento do DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node query.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Excluir um item de uma tabela

Crie um módulo do Node.js com o nome de arquivo `delete.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Isso inclui `@aws-sdk/lib-dynamodb`, um pacote de biblioteca que fornece funcionalidade de cliente de documentos para `@aws-sdk/client-dynamodb`. Em seguida, defina a configuração conforme mostrado abaixo para realizar marshalling e unmarshalling - como um segundo parâmetro opcional - durante a criação do cliente do documento. Em seguida, crie os clientes. Para acessar o DynamoDB, crie um objeto `DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para excluir um item na tabela, que, neste exemplo, inclui o nome da tabela, bem como o nome e o valor da chave hash do item que você deseja excluir. Chame o método `DeleteCommand` do cliente de documento do DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
```

```
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node delete.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do AWS Elemental MediaConvert

O AWS Elemental MediaConvert é um serviço de transcodificação de vídeo baseado em arquivo com recursos de nível de transmissão. Você pode usá-lo para criar ativos para fornecer transmissão e vídeo sob demanda (VoD) na Internet. Para obter mais informações, consulte o Guia do usuário do AWS Elemental MediaConvert.

A API JavaScript para MediaConvert é exposta por meio da classe de cliente do `MediaConvert`. Para obter mais informações, consulte Classe: MediaConvert na Referência da API.

Tópicos

- Como obter seu endpoint específico da região para o MediaConvert
- Criar e gerenciar tarefas de transcodificação no MediaConvert
- Uso de modelos de trabalho no MediaConvert

Como obter seu endpoint específico da região para o MediaConvert

Este exemplo de código Node.js mostra:

- Como recuperar o endpoint específico da região do MediaConvert.

O cenário

Neste exemplo, você usa um módulo Node.js para chamar o MediaConvert e recuperar o endpoint específico da região. Você pode recuperar o URL do seu endpoint a partir do endpoint padrão de serviço e, dessa forma, o endpoint específico por região ainda não será necessário. O código usa

o SDK para JavaScript para recuperar esse endpoint usando este método da classe de cliente do MediaConvert:

- `DescribeEndpointsCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.
- Crie um perfil do IAM que dê ao MediaConvert acesso aos arquivos de entrada e aos buckets do Amazon S3 onde os arquivos de saída são armazenados. Para obter detalhes, consulte Configurar permissões do IAM no Guia do usuário do AWS Elemental MediaConvert.

> **Important**
>
> Este exemplo usa ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Obter o URL do endpoint

Crie um diretório `libs` e um módulo Node.js com o nome do arquivo`emcClientGet.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do MediaConvert. Substitua *`REGION`* pela sua região da AWS.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the AWS Region.
const REGION = "REGION";
//Set the MediaConvert Service Object
const emcClientGet = new MediaConvertClient({ region: REGION });
export { emcClientGet };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `emc_getendpoint.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto a fim de passar os parâmetros de solicitação vazios para o método `DescribeEndpointsCommand` da classe de cliente do MediaConvert. Depois, chame o método `DescribeEndpointsCommand`.

```
// Import required AWS-SDK clients and commands for Node.js
import { DescribeEndpointsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClientGet } from "./libs/emcClientGet.js";

//set the parameters.
const params = { MaxResults: 0 };

const run = async () => {
  try {
    // Create a new service object and set MediaConvert to customer endpoint
    const data = await emcClientGet.send(new DescribeEndpointsCommand(params));
    console.log("Your MediaConvert endpoint is ", data.Endpoints);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node emc_getendpoint.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Criar e gerenciar tarefas de transcodificação no MediaConvert

Este exemplo de código Node.js mostra:

- Como especificar o endpoint específico por região a ser usado com o MediaConvert.
- Como criar tarefas de transcodificação no MediaConvert.
- Como cancelar uma tarefa de transcodificação.
- Como recuperar o JSON para concluir uma tarefa de transcodificação.
- Como recuperar uma matriz JSON para até 20 das tarefas criadas mais recentemente.

O cenário

Neste exemplo, você usa um módulo Node.js a fim de chamar o MediaConvert para criar e gerenciar tarefas de transcodificação. O código usa o SDK para JavaScript para fazer isso usando estes métodos da classe de cliente do MediaConvert:

- `CreateJobCommand`
- `CancelJobCommand`
- `GetJobCommand`
- `ListJobsCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.
- Crie e configure buckets do Amazon S3 que forneçam armazenamento para arquivos de entrada e de saída da tarefa. Para obter detalhes, consulte Criar armazenamento para arquivos no Guia do usuário do AWS Elemental MediaConvert.
- Faça upload do vídeo de entrada no bucket do Amazon S3 provisionado para armazenamento de entrada. Para obter uma lista dos codecs de vídeo de entrada e contêineres compatíveis, consulte Codecs e contêineres de entrada compatíveis no Guia do usuário do AWS Elemental MediaConvert.

- Crie um perfil do IAM que dê ao MediaConvert acesso aos arquivos de entrada e aos buckets do Amazon S3 onde os arquivos de saída são armazenados. Para obter detalhes, consulte Configurar permissões do IAM no Guia do usuário do AWS Elemental MediaConvert.

> **Important**
>
> Este exemplo usa ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Como configurar o SDK

Configure o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Como o MediaConvert usa endpoints personalizados para cada conta, você também deve configurar a classe de cliente do `MediaConvert` para usar o endpoint específico da região. Para isso, defina o parâmetro `endpoint` em `mediaconvert(endpoint)`.

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";
```

Definição de uma tarefa de transcodificação simples

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`emcClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do MediaConvert. Substitua *`REGION`* pela sua região da AWS. Substitua *`ENDPOINT`* pelo endpoint da sua conta do MediaConvert, o que você pode fazer na página Conta no console do MediaConvert.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `emc_createjob.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie o JSON que define os parâmetros da tarefa de transcodificação.

Esses parâmetros são bem detalhados. Você pode usar o console do AWS Elemental MediaConvert para gerar os parâmetros de trabalho do JSON escolhendo as configurações de tarefa no console e depois selecionando Mostrar JSON de trabalho na parte inferior da seção Trabalho. Este exemplo mostra o JSON para uma tarefa simples.

> **Note**
>
> *Substitua JOB_QUEUE_ARN pela fila de trabalhos do MediaConvert, IAM_ROLE_ARN pelo nome do recurso da Amazon (ARN) do perfil do IAM, OUTPUT_BUCKET_NAME pelo nome do bucket de destino, por exemplo, "s3://OUTPUT_BUCKET_NAME/", e INPUT_BUCKET_AND_FILENAME* pelo bucket de entrada e nome do arquivo, por exemplo, "s3://INPUT_BUCKET/FILE_NAME".

```
const params = {
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN", //IAM_ROLE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "OUTPUT_BUCKET_NAME", //OUTPUT_BUCKET_NAME, e.g., "s3://
BUCKET_NAME/"
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
```

```
            TimecodeInsertion: "DISABLED",
            AntiAlias: "ENABLED",
            Sharpness: 50,
            CodecSettings: {
              Codec: "H_264",
              H264Settings: {
                InterlaceMode: "PROGRESSIVE",
                NumberReferenceFrames: 3,
                Syntax: "DEFAULT",
                Softness: 0,
                GopClosedCadence: 1,
                GopSize: 90,
                Slices: 1,
                GopBReference: "DISABLED",
                SlowPal: "DISABLED",
                SpatialAdaptiveQuantization: "ENABLED",
                TemporalAdaptiveQuantization: "ENABLED",
                FlickerAdaptiveQuantization: "DISABLED",
                EntropyEncoding: "CABAC",
                Bitrate: 5000000,
                FramerateControl: "SPECIFIED",
                RateControlMode: "CBR",
                CodecProfile: "MAIN",
                Telecine: "NONE",
                MinIInterval: 0,
                AdaptiveQuantization: "HIGH",
                CodecLevel: "AUTO",
                FieldEncoding: "PAFF",
                SceneChangeDetect: "ENABLED",
                QualityTuningLevel: "SINGLE_PASS",
                FramerateConversionAlgorithm: "DUPLICATE_DROP",
                UnregisteredSeiTimecode: "DISABLED",
                GopSizeUnits: "FRAMES",
                ParControl: "SPECIFIED",
                NumberBFramesBetweenReferenceFrames: 2,
                RepeatPps: "DISABLED",
                FramerateNumerator: 30,
                FramerateDenominator: 1,
                ParNumerator: 1,
                ParDenominator: 1,
              },
            },
            AfdSignaling: "NONE",
            DropFrameTimecode: "ENABLED",
```

```
            RespondToAfd: "NONE",
            ColorMetadata: "INSERT",
          },
          AudioDescriptions: [
            {
              AudioTypeControl: "FOLLOW_INPUT",
              CodecSettings: {
                Codec: "AAC",
                AacSettings: {
                  AudioDescriptionBroadcasterMix: "NORMAL",
                  RateControlMode: "CBR",
                  CodecProfile: "LC",
                  CodingMode: "CODING_MODE_2_0",
                  RawFormat: "NONE",
                  SampleRate: 48000,
                  Specification: "MPEG4",
                  Bitrate: 64000,
                },
              },
              LanguageCodeControl: "FOLLOW_INPUT",
              AudioSourceName: "Audio Selector 1",
            },
          ],
          ContainerSettings: {
            Container: "MP4",
            Mp4Settings: {
              CslgAtom: "INCLUDE",
              FreeSpaceBox: "EXCLUDE",
              MoovPlacement: "PROGRESSIVE_DOWNLOAD",
            },
          },
          NameModifier: "_1",
        },
      ],
    },
  ],
  AdAvailOffset: 0,
  Inputs: [
    {
      AudioSelectors: {
        "Audio Selector 1": {
          Offset: 0,
          DefaultSelection: "NOT_DEFAULT",
          ProgramSelection: 1,
```

```
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
 "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};
```

Criar uma tarefa de transcodificação

Depois de criar o JSON de parâmetros de tarefa, chame o método `run` assíncrono para invocar um objeto de serviço de cliente do `MediaConvert`, passando os parâmetros. O ID da tarefa criado é retornado no `data` da resposta.

```
const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Job created!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node emc_createjob.js
```

Esse código de exemplo completo pode ser encontrado aqui no GitHub.

Cancelar uma tarefa de transcodificação

Crie um diretório `libs` e um módulo Node.js com o nome do arquivo`emcClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do MediaConvert. Substitua *`REGION`* pela sua região da AWS. Substitua *`ENDPOINT`* pelo endpoint da sua conta do MediaConvert, o que você pode fazer na página Conta do console do MediaConvert.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `emc_canceljob.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Crie o JSON que inclua o ID da tarefa a ser cancelada. Depois, chame o método `CancelJobCommand` criando uma promessa para invocar um objeto de serviço de cliente do `MediaConvert`, passando os parâmetros. Lide com a resposta no retorno de chamada da promessa.

> **Note**
>
> Substitua *`JOB_ID`* pelo ID da tarefa a ser cancelada.

```
// Import required AWS-SDK clients and commands for Node.js
import { CancelJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = { Id: "JOB_ID" }; //JOB_ID
```

```
const run = async () => {
  try {
    const data = await emcClient.send(new CancelJobCommand(params));
    console.log("Job  " + params.Id + " is canceled");
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node ec2_canceljob.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Como lista tarefas de transcodificação recentes

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`emcClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do MediaConvert. Substitua *`REGION`* pela sua região da AWS. Substitua *`ENDPOINT`* pelo endpoint da sua conta do MediaConvert, o que você pode fazer na página Conta do console do MediaConvert.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `emc_listjobs.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie o JSON de parâmetros, incluindo valores para especificar se você deseja classificar a lista em ordem `ASCENDING` ou `DESCENDING`, o Nome do recurso da Amazon (ARN) da fila de tarefas a ser

verificada e o status das tarefas a serem incluídas. Depois, chame o método `ListJobsCommand` criando uma promessa para invocar um objeto de serviço de cliente do `MediaConvert`, passando os parâmetros.

> **Note**
>
> Substitua *`QUEUE_ARN`* pelo Nome do recurso da Amazon (ARN) da fila de tarefas a ser verificada e *`STATUS`* pelo status da fila.

```
// Import required AWS-SDK clients and commands for Node.js
import { ListJobsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED", // e.g., "SUBMITTED"
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobsCommand(params));
    console.log("Success. Jobs: ", data.Jobs);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node emc_listjobs.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Uso de modelos de trabalho no MediaConvert

Este exemplo de código Node.js mostra:

- Como criar modelos de tarefa do AWS Elemental MediaConvert.
- Como usar um modelo de tarefa para criar uma tarefa de transcodificação.
- Como listar todos os modelos de tarefa.
- Como excluir modelos de tarefa.

O cenário

O JSON necessário para criar um trabalho de transcodificação no MediaConvert é detalhado, contendo um grande número de configurações. Simplifique muito a criação de tarefas salvando as configurações em boas condições em um modelo de tarefa que você possa usar para criar tarefas subsequentes. Neste exemplo, você usa um módulo Node.js a fim de chamar o MediaConvert para criar, usar e gerenciar modelos de trabalho. O código usa o SDK para JavaScript para fazer isso usando estes métodos da classe de cliente do MediaConvert:

- `CreateJobTemplateCommand`
- `CreateJobCommand`
- `DeleteJobTemplateCommand`
- `ListJobTemplatesCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

- Crie um perfil do IAM que dê ao MediaConvert acesso aos arquivos de entrada e aos buckets do Amazon S3 onde os arquivos de saída são armazenados. Para obter detalhes, consulte Configurar permissões do IAM no Guia do usuário do AWS Elemental MediaConvert.

> **Important**
>
> Esses exemplos usam ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js. No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Criar um modelo de trabalho

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`emcClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do MediaConvert. Substitua *`REGION`* pela sua região da AWS. Substitua *`ENDPOINT`* pelo endpoint da sua conta do MediaConvert, o que você pode fazer na página Conta do console do MediaConvert.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `emc_create_jobtemplate.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Especifique o JSON de parâmetros para a criação do modelo. Use a maioria dos parâmetros JSON de uma tarefa anterior bem-sucedida para especificar os valores `Settings` no modelo. Este exemplo usa as configurações de tarefa de Criar e gerenciar tarefas de transcodificação no MediaConvert.

Chame o método `CreateJobTemplateCommand` criando uma promessa para invocar um objeto de serviço de cliente do `MediaConvert`, passando os parâmetros.

> **Note**
>
> Substitua *`JOB_QUEUE_ARN`* pelo Nome do recurso da Amazon (ARN) da fila de trabalhos a ser verificada e *`BUCKET_NAME`* pelo nome do bucket do Amazon S3 de destino. Por exemplo, "s3://BUCKET_NAME/".

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  Category: "YouTube Jobs",
  Description: "Final production transcode",
  Name: "DemoTemplate",
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "BUCKET_NAME", // BUCKET_NAME e.g., "s3://BUCKET_NAME/"
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
              TimecodeInsertion: "DISABLED",
              AntiAlias: "ENABLED",
              Sharpness: 50,
              CodecSettings: {
                Codec: "H_264",
                H264Settings: {
                  InterlaceMode: "PROGRESSIVE",
                  NumberReferenceFrames: 3,
                  Syntax: "DEFAULT",
```

```
            Softness: 0,
            GopClosedCadence: 1,
            GopSize: 90,
            Slices: 1,
            GopBReference: "DISABLED",
            SlowPal: "DISABLED",
            SpatialAdaptiveQuantization: "ENABLED",
            TemporalAdaptiveQuantization: "ENABLED",
            FlickerAdaptiveQuantization: "DISABLED",
            EntropyEncoding: "CABAC",
            Bitrate: 5000000,
            FramerateControl: "SPECIFIED",
            RateControlMode: "CBR",
            CodecProfile: "MAIN",
            Telecine: "NONE",
            MinIInterval: 0,
            AdaptiveQuantization: "HIGH",
            CodecLevel: "AUTO",
            FieldEncoding: "PAFF",
            SceneChangeDetect: "ENABLED",
            QualityTuningLevel: "SINGLE_PASS",
            FramerateConversionAlgorithm: "DUPLICATE_DROP",
            UnregisteredSeiTimecode: "DISABLED",
            GopSizeUnits: "FRAMES",
            ParControl: "SPECIFIED",
            NumberBFramesBetweenReferenceFrames: 2,
            RepeatPps: "DISABLED",
            FramerateNumerator: 30,
            FramerateDenominator: 1,
            ParNumerator: 1,
            ParDenominator: 1,
          },
        },
        AfdSignaling: "NONE",
        DropFrameTimecode: "ENABLED",
        RespondToAfd: "NONE",
        ColorMetadata: "INSERT",
      },
      AudioDescriptions: [
        {
          AudioTypeControl: "FOLLOW_INPUT",
          CodecSettings: {
            Codec: "AAC",
            AacSettings: {
```

```
                AudioDescriptionBroadcasterMix: "NORMAL",
                RateControlMode: "CBR",
                CodecProfile: "LC",
                CodingMode: "CODING_MODE_2_0",
                RawFormat: "NONE",
                SampleRate: 48000,
                Specification: "MPEG4",
                Bitrate: 64000,
              },
            },
            LanguageCodeControl: "FOLLOW_INPUT",
            AudioSourceName: "Audio Selector 1",
          },
        ],
        ContainerSettings: {
          Container: "MP4",
          Mp4Settings: {
            CslgAtom: "INCLUDE",
            FreeSpaceBox: "EXCLUDE",
            MoovPlacement: "PROGRESSIVE_DOWNLOAD",
          },
        },
        NameModifier: "_1",
      },
    ],
  },
],
AdAvailOffset: 0,
Inputs: [
  {
    AudioSelectors: {
      "Audio Selector 1": {
        Offset: 0,
        DefaultSelection: "NOT_DEFAULT",
        ProgramSelection: 1,
        SelectorType: "TRACK",
        Tracks: [1],
      },
    },
    VideoSelector: {
      ColorSpace: "FOLLOW",
    },
    FilterEnable: "AUTO",
    PsiControl: "USE_PSI",
```

```
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};

const run = async () => {
  try {
    // Create a promise on a MediaConvert object
    const data = await emcClient.send(new CreateJobTemplateCommand(params));
    console.log("Success!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node emc_create_jobtemplate.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Criar um trabalho de transcodificação com base em um modelo de trabalho

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`emcClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do MediaConvert. Substitua *`REGION`* pela sua região da AWS. Substitua *`ENDPOINT`* pelo endpoint da sua conta do MediaConvert, o que você pode fazer na página Conta do console do MediaConvert.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
```

```
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `emc_template_createjob.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie o JSON de parâmetros da criação de tarefas, inclusive o nome do modelo de tarefa a ser usado, e o `Settings` para usar os específicos da tarefa que você está criando. Depois, chame o método `CreateJobsCommand` criando uma promessa para invocar um objeto de serviço de cliente do `MediaConvert`, passando os parâmetros.

> **Note**
>
> Substitua *`JOB_QUEUE_ARN`* pelo Nome do recurso da Amazon (ARN) da fila de trabalhos a ser verificada, *`KEY_PAIR_NAME`*, *`TEMPLATE_NAME`* e *`ROLE_ARN`* pelo Nome do recurso da Amazon (ARN) da função e *`INPUT_BUCKET_AND_FILENAME`* pelo bucket de entrada e nome do arquivo. Por exemplo, "s3://BUCKET_NAME/FILE_NAME".

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  Queue: "QUEUE_ARN", //QUEUE_ARN
  JobTemplate: "TEMPLATE_NAME", //TEMPLATE_NAME
  Role: "ROLE_ARN", //ROLE_ARN
  Settings: {
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
```

```
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
 "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
  },
};

const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Success! ", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node emc_template_createjob.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Como listar os modelos de trabalho

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`emcClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do MediaConvert. Substitua *`REGION`* pela sua região da AWS. Substitua *`ENDPOINT`* pelo endpoint da sua conta do MediaConvert, o que você pode fazer na página Conta do console do MediaConvert.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
```

```
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `emc_listtemplates.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar os parâmetros de solicitação para o método `listTemplates` da classe de cliente `MediaConvert`. Inclua valores para determinar quais modelos listar (`NAME`, `CREATION_DATE`, `SYSTEM`), quantas listar e a ordem de classificação. Para chamar o método `ListTemplatesCommand`, crie uma promessa para invocar um objeto de serviço de cliente do MediaConvert, passando os parâmetros.

```
// Import required AWS-SDK clients and commands for Node.js
import { ListJobTemplatesCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  ListBy: "NAME",
  MaxResults: 10,
  Order: "ASCENDING",
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobTemplatesCommand(params));
    console.log("Success ", data.JobTemplates);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node emc_listtemplates.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Excluir um modelo de trabalho

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`emcClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do MediaConvert. Substitua *`REGION`* pela sua região da AWS. Substitua *`ENDPOINT`* pelo endpoint da sua conta do MediaConvert, o que você pode fazer na página Conta do console do MediaConvert.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `emc_deletetemplate.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar o nome do modelo de tarefa que você deseja excluir como parâmetro do método `DeleteJobTemplateCommand` da classe de cliente `MediaConvert`. Para chamar o método `DeleteJobTemplateCommand`, crie uma promessa para invocar um objeto de serviço de cliente do MediaConvert, passando os parâmetros.

```
// Import required AWS-SDK clients and commands for Node.js
import { DeleteJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = { Name: "test" }; //TEMPLATE_NAME

const run = async () => {
  try {
    const data = await emcClient.send(new DeleteJobTemplateCommand(params));
```

```
    console.log(
      "Success, template deleted! Request ID:",
      data.$metadata.requestId,
    );
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node emc_deletetemplate.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do AWS Lambda

AWS Lambda é um serviço de computação com tecnologia sem servidor que permite executar código sem provisionar ou gerenciar servidores, criar uma lógica de escalonamento de cluster com reconhecimento de workload, manter integrações de eventos ou gerenciar runtimes.

A API JavaScript do AWS Lambda é exposta por meio da classe de cliente LambdaService.

Veja a seguir uma lista de exemplos que demonstram como criar e usar funções do Lambda com o AWS SDK for JavaScript v3:

- Invocar o Lambda com o API Gateway
- Criação de eventos programados para executar funções do AWS Lambda

Exemplos do Amazon Lex

O Amazon Lex é um serviço da AWS para a criação de interfaces de conversa em aplicativos que usam voz e texto.

A API JavaScript para Amazon Lex é exposta por meio da classe de cliente Serviço de runtime do Lex.

- Como criar um chatbot do Amazon Lex

Exemplos do Amazon Polly

Este exemplo de código Node.js mostra:

- Faça upload do áudio gravado usando o Amazon Polly para o Amazon S3

O cenário

Neste exemplo, uma série de módulos Node.js é usada para carregar automaticamente o áudio gravado usando o Amazon Polly para o Amazon S3 usando esses métodos da classe de cliente Amazon S3:

- `StartSpeechSynthesisTaskCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure um ambiente de projeto para executar JavaScript exemplos do Node seguindo as instruções em GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS .
- Crie uma política de funções de usuário do Amazon Cognito AWS Identity and Access Management (IAM) não autenticadaSynthesizeSpeech : permissões e um pool de identidade do Amazon Cognito com a função do IAM anexada a ela. A seção Crie os AWS recursos usando o AWS CloudFormation abaixo descreve como criar esses recursos.

> **Note**
>
> Este exemplo usa o Amazon Cognito, mas se você não estiver usando o Amazon Cognito, AWS seu usuário deverá seguir a política de permissões do IAM

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "mobileanalytics:PutEvents",
        "cognito-sync:*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "polly:SynthesizeSpeech",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Crie os AWS recursos usando o AWS CloudFormation

AWS CloudFormation permite que você crie e provisione implantações de AWS infraestrutura de forma previsível e repetida. Para obter mais informações sobre AWS CloudFormation, consulte o Guia AWS CloudFormation do usuário.

Para criar a AWS CloudFormation pilha:

1. Instale e configure as instruções a AWS CLI seguir no Guia AWS CLI do usuário.
2. Crie um arquivo chamado `setup.yaml` no diretório raiz da pasta do seu projeto e copie o conteúdo aqui GitHub para dentro dele.

> **Note**
>
> O AWS CloudFormation modelo foi gerado usando o AWS CDK disponível aqui em GitHub. Para obter mais informações sobre o AWS CDK, consulte o Guia do AWS Cloud Development Kit (AWS CDK) desenvolvedor.

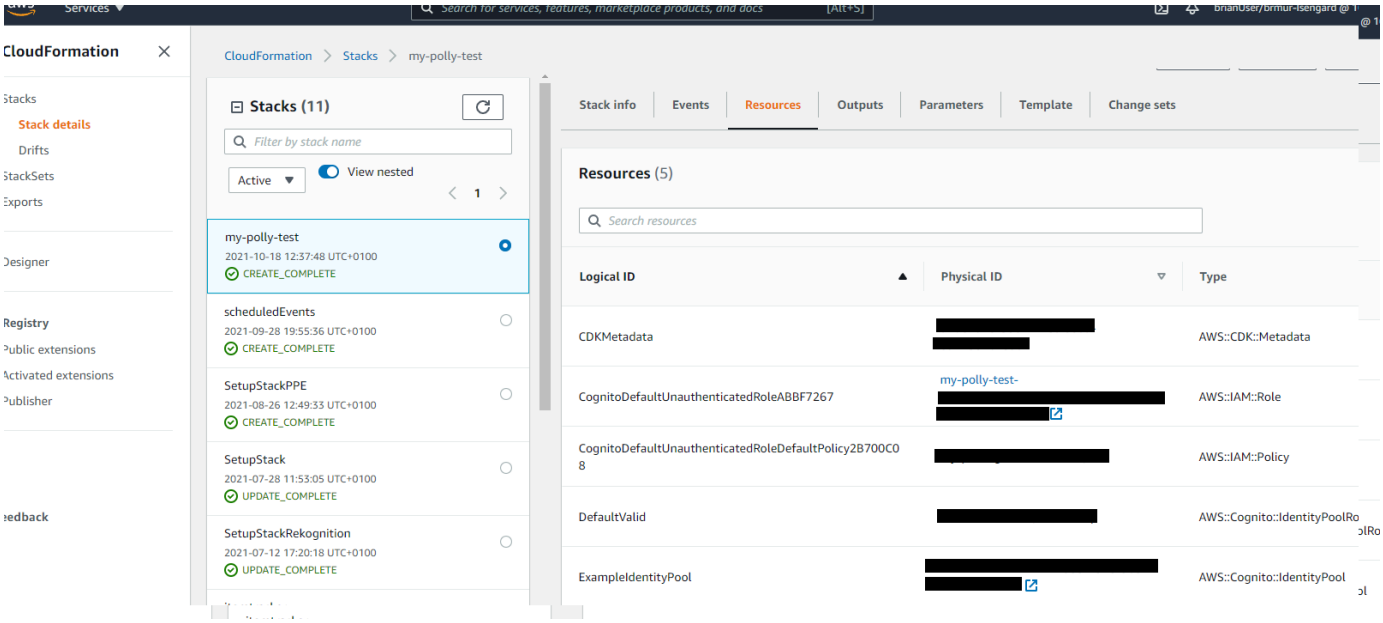
3. Execute o comando a seguir na linha de comando, substituindo *STACK_NAME* por um nome exclusivo para a pilha.

> **Important**
>
> O nome da pilha deve ser exclusivo dentro de uma AWS região e AWS conta. Você pode especificar até 128 caracteres. São permitidos números e hifens.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

Para obter mais informações sobre os parâmetros do comando `create-stack`, consulte o Guia de referência de comandos da AWS CLI e o Guia do usuário do AWS CloudFormation.

4. Navegue até o console AWS CloudFormation de gerenciamento, escolha Pilhas, escolha o nome da pilha e escolha a guia Recursos para ver uma lista dos recursos criados.



Faça upload do áudio gravado usando o Amazon Polly para o Amazon S3

Crie um módulo do Node.js com o nome de arquivo `polly_synthesize_to_s3.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. No código, insira a *REGION* e o *BUCKET_NAME*. Para acessar o Amazon Polly, crie um objeto de serviço de cliente do `Polly`. Substitua *"IDENTITY_POOL_ID"* pelo `IdentityPoolId` da

página de exemplo do banco de identidades do Amazon Cognito que você criou para este exemplo. Isso também é passado para cada objeto do cliente.

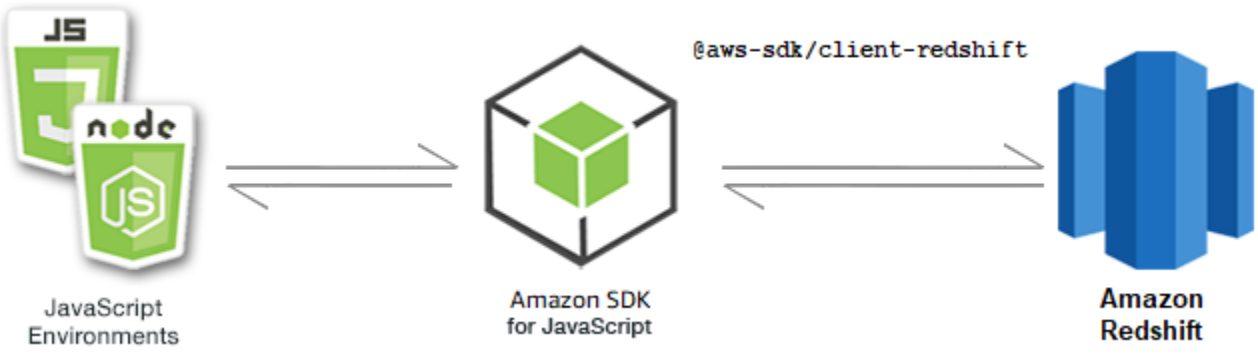
Chame o método `StartSpeechSynthesisCommand` do objeto de serviço de cliente do Amazon Polly, sintetize a mensagem de voz e carregue-a para o bucket do Amazon S3.

```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "./libs/pollyClient.js";

// Create the parameters
var params = {
  OutputFormat: "mp3",
  OutputS3BucketName: "videoanalyzerbucket",
  Text: "Hello David, How are you?",
  TextType: "text",
  VoiceId: "Joanna",
  SampleRate: "22050",
};

const run = async () => {
  try {
    await pollyClient.send(new StartSpeechSynthesisTaskCommand(params));
    console.log("Success, audio file added to " + params.OutputS3BucketName);
  } catch (err) {
    console.log("Error putting object", err);
  }
};
run();
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Exemplos do Amazon Redshift

O Amazon Redshift é um serviço de data warehouse em escala de petabytes totalmente gerenciado na nuvem. Um data warehouse do Amazon Redshift é um conjunto de recursos de computação chamados nós, que são organizados em um grupo chamado cluster. Cada cluster executa um mecanismo do Amazon Redshift e contém um ou mais bancos de dados.



A API JavaScript para Amazon Redshift é exposta por meio da classe de cliente do Amazon Redshift.

Tópicos

- Exemplos do Amazon Redshift

Exemplos do Amazon Redshift

Neste exemplo, uma série de módulos Node.js é usada para criar, modificar, descrever os parâmetros e, em seguida, excluir clusters do Amazon Redshift usando os seguintes métodos da classe de cliente do `Redshift`:

- `CreateClusterCommand`
- `ModifyClusterCommand`
- `DescribeClustersCommand`
- `DeleteClusterCommand`

Para obter mais informações sobre usuários do Amazon Redshift, consulte o Guia de conceitos básicos do Amazon Redshift.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Criação de um cluster do Amazon Redshift

Este exemplo demonstra como criar um cluster do Amazon Redshift usando o AWS SDK for JavaScript. Para obter mais informações, consulte CreateCluster.

> **Important**
>
> O cluster que você está prestes a criar está ativo (e não está sendo executado em uma sandbox). Você será cobrado pelas taxas de uso padrão do Amazon Redshift para o cluster até que o exclua. Se você excluir o cluster na mesma sessão em que o criou, o valor total cobrado será mínimo.

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`redshiftClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Redshift. Substitua *`REGION`* pela sua região da AWS.

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `redshift-create-cluster.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie um objeto de parâmetros, especificando o tipo de nó a ser provisionado, e as credenciais principais de login para a instância do banco de dados criada automaticamente no cluster e, finalmente, o tipo de cluster.

> **Note**
>
> Substitua *`CLUSTER_NAME`* pelo nome do cluster. Para *`NODE_TYPE`*, especifique o tipo de nó a ser provisionado, como 'dc2.large', por exemplo. *`MASTER_USERNAME`* e *`MASTER_USER_PASSWORD`* são as credenciais de login do usuário principal da sua instância de banco de dados no cluster. Para *`CLUSTER_TYPE`*, insira o tipo de cluster. Se você especificar `single-node`, não precisará do parâmetro `NumberOfNodes`. Todos os parâmetros restantes são opcionais.

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least one
 uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
 your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
 cluster subnet group to be associated with this cluster. Defaults to 'default' if not
 specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
```

```
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node redshift-create-cluster.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Modificação de um cluster do Amazon Redshift

Este exemplo mostra como modificar a senha do usuário principal de um cluster do Amazon Redshift usando o AWS SDK for JavaScript. Para obter mais informações sobre quais outras configurações você pode modificar, consulte ModifyCluster.

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`redshiftClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Redshift. Substitua *`REGION`* pela sua região da AWS.

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `redshift-modify-cluster.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Especifique a Região AWS, o nome do cluster que você deseja modificar e a nova senha do usuário principal.

> **Note**
>
> Substitua *CLUSTER_NAME* pelo nome do cluster e *MASTER_USER_PASSWORD* pela nova senha do usuário principal.

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node redshift-modify-cluster.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Visualização de detalhes de um cluster do Amazon Redshift

Este exemplo mostra como visualizar os detalhes de um cluster do Amazon Redshift usando o AWS SDK for JavaScript. Para obter mais informações sobre opcionais, consulte DescribeClusters.

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`redshiftClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Redshift. Substitua *REGION* pela sua região da AWS.

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `redshift-describe-clusters.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Especifique a Região AWS, o nome do cluster que você deseja modificar e a nova senha do usuário principal.

> **Note**
>
> Substitua *CLUSTER_NAME* pelo nome do cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node redshift-describe-clusters.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um cluster do Amazon Redshift

Este exemplo mostra como visualizar os detalhes de um cluster do Amazon Redshift usando o AWS SDK for JavaScript. Para obter mais informações sobre quais outras configurações você pode modificar, consulte DeleteCluster.

Crie um diretório `libs` e um módulo Node.js com o nome de arquivo`redshiftClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Redshift. Substitua *`REGION`* pela sua região da AWS.

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo Node.js com o nome de arquivo `redshift-delete-clusters.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Especifique a Região AWS, o nome do cluster que você deseja modificar e a nova senha do usuário principal. Em seguida, especifique se você deseja salvar um snapshot final do cluster antes de excluí-lo e, em caso afirmativo, o ID do snapshot.

> **Note**
>
> Substitua *`CLUSTER_NAME`* pelo nome do cluster. Para o *`SkipFinalClusterSnapshot`*, especifique se deseja criar um snapshot final do cluster antes de excluí-lo. Se você especificar 'false', especifique o id do snapshot final do cluster em *`CLUSTER_SNAPSHOT_ID`*. Você pode obter esse ID clicando no link da coluna Snapshots do cluster no painel Clusters e rolando para baixo até o painel Snapshots. Observe que o radical `rs:` não faz parte do ID do snapshot.

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

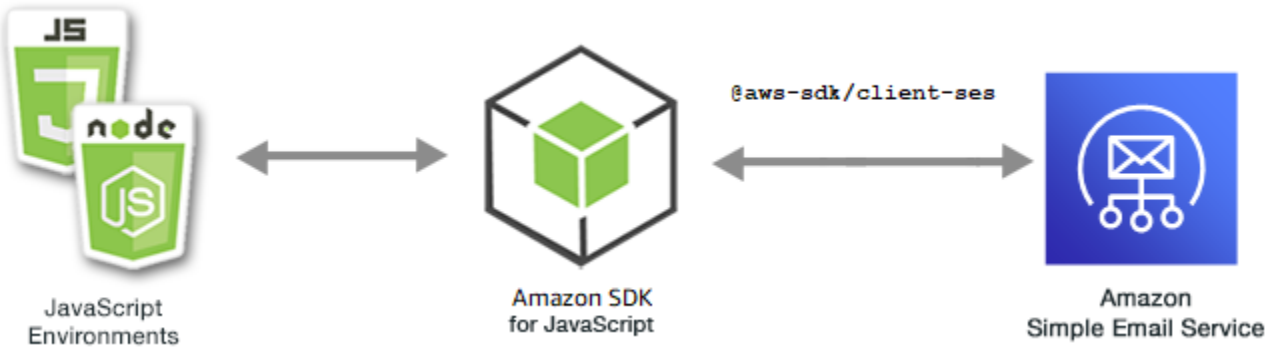
Para executar o exemplo, digite o seguinte no prompt de comando.

```
node redshift-delete-cluster.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do Amazon Simple Email Service

O Amazon Simple Email Service (Amazon SES) é um serviço de envio de e-mails baseado na nuvem criado para ajudar profissionais de marketing digital e desenvolvedores de aplicativos a enviar e-mails de marketing, notificações e mensagens transacionais. Ele é um serviço confiável e econômico para empresas de todos os tamanhos que usam e-mail para manter contato com seus clientes.



A JavaScript API do Amazon SES é exposta por meio da classe SES cliente. Para obter mais informações sobre como usar a classe de cliente do Amazon SES, consulte Classe: SES na Referência da API.

Tópicos

- Gerenciamento de identidades do Amazon SES
- Trabalhar com modelos de e-mail no Amazon SES
- Envio de e-mail usando o Amazon SES

Gerenciamento de identidades do Amazon SES

Este exemplo de código Node.js mostra:

- Como verificar endereços de e-mail e domínios usados com o Amazon SES.
- Como atribuir uma política AWS Identity and Access Management (IAM) às suas identidades do Amazon SES.
- Como listar todas as identidades do Amazon SES para sua AWS conta.
- Como excluir identidades usadas com o Amazon SES.

Uma identidade do Amazon SES é um endereço de e-mail ou domínio que o Amazon SES usa para enviar e-mails. O Amazon SES requer que você verifique suas identidades de e-mail, confirmando que você é o proprietário delas e impedindo que outras pessoas as utilizem.

Para obter detalhes sobre como verificar endereços de e-mail e domínios no Amazon SES, consulte Verificar endereços de e-mail e domínios no Amazon SES no Guia do desenvolvedor do Amazon Simple Email Service. Para obter informações sobre autorização de envio no Amazon SES, consulte Visão geral da autorização de envio do Amazon SES.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para verificar e gerenciar identidades do Amazon SES. Os módulos Node.js usam o SDK JavaScript para verificar endereços de e-mail e domínios, usando esses métodos da classe SES cliente:

- `ListIdentitiesCommand`
- `DeleteIdentityCommand`
- `VerifyEmailIdentityCommand`
- `VerifyDomainIdentityCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses TypeScript exemplos de Node e instale os módulos necessários AWS SDK for JavaScript e de terceiros. Siga as instruções em GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS .

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Listar suas identidades

Neste exemplo, use um módulo do Node.js para listar endereços de e-mail e domínios para usar com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_listidentities.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar o `IdentityType` e outros parâmetros para o método `ListIdentitiesCommand` da classe de cliente `SES`. Para chamar o método `ListIdentitiesCommand`, invoque um objeto de serviço do Amazon SES passando o objeto dos parâmetros.

O `data` retornado contém um array de identidades de domínio, conforme especificado pelo parâmetro `IdentityType`.

> **Note**
>
> Substitua *`IDENTITY_TYPE`* pelo tipo de identidade, que pode ser "EmailAddress" ou “Domínio”.

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });
```

```
const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node ses_listidentities.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Verificar a identidade de um endereço de e-mail

Neste exemplo, use um módulo do Node.js para verificar remetentes de e-mail para usar com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_verifyemailidentity.js`. Configure o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários.

Crie um objeto para passar o parâmetro `EmailAddress` para o método `VerifyEmailIdentityCommand` da classe de cliente `SES`. Para chamar o método `VerifyEmailIdentityCommand`, invoque um objeto de serviço de cliente do Amazon SES, passando os parâmetros.

> **Note**
>
> Substitua *ADDRESS@DOMAIN.EXT* pelo endereço de e-mail, como nome@exemplo.com.

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O domínio é adicionado ao Amazon SES a ser verificado.

```
node ses_verifyemailidentity.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Verificar uma identidade de domínio

Neste exemplo, use um módulo do Node.js para verificar domínios de e-mail a serem usados com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *REGION* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_verifydomainidentity.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar o parâmetro `Domain` para o método `VerifyDomainIdentityCommand` da classe de cliente `SES`. Para chamar o método `VerifyDomainIdentityCommand`, invoque um objeto de serviço de cliente do Amazon SES, passando o objeto dos parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *`AMI_ID`* pelo ID da imagem de máquina da Amazon (AMI) a ser executada, e *`KEY_PAIR_NAME`* do par de chaves a ser atribuído ao ID da AMI.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
```

```
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O domínio é adicionado ao Amazon SES a ser verificado.

```
node ses_verifydomainidentity.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Excluir identidades

Neste exemplo, use um módulo do Node.js para excluir endereços de e-mail ou domínios usados com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_deleteidentity.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar o parâmetro `Identity` para o método `DeleteIdentityCommand` da classe de cliente `SES`. Para chamar o método `DeleteIdentityCommand`, crie uma `request` para invocar um objeto de serviço de cliente do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *`IDENTITY_TYPE`* pelo tipo de identidade a ser excluído e *`IDENTITY_NAME`* pelo nome da identidade a ser excluída.

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
```

```
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node ses_deleteidentity.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Trabalhar com modelos de e-mail no Amazon SES

Este exemplo de código Node.js mostra:

- Como obter uma lista de todos os modelos de e-mail.
- Como recuperar e atualizar os modelos de e-mail.
- Como criar e excluir modelos de e-mail.

O Amazon SES permite que você envie mensagens de e-mail personalizadas usando modelos de e-mail. Para obter detalhes sobre como criar e usar modelos de e-mail no Amazon SES, consulte Envio de e-mail personalizado usando a API do Amazon SES no Guia do desenvolvedor do Amazon Simple Email Service.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para trabalhar com modelos de e-mail. Os módulos Node.js usam o SDK JavaScript para criar e usar modelos de e-mail usando esses métodos da classe SES cliente:

- `ListTemplatesCommand`
- `CreateTemplateCommand`
- `GetTemplateCommand`
- `DeleteTemplateCommand`

- `UpdateTemplateCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses TypeScript exemplos de Node e instale os módulos necessários AWS SDK for JavaScript e de terceiros. Siga as instruções em GitHub.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS .

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Listar seus modelos de e-mail

Neste exemplo, use um módulo do Node.js para criar um modelo de e-mail a ser usado com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
```

```
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_listtemplates.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar os parâmetros para o método `ListTemplatesCommand` da classe de cliente `SES`. Para chamar o método `ListTemplatesCommand`, invoque um objeto de serviço de cliente do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *`ITEMS_COUNT`* pelo número máximo de modelos a serem retornados. O valor deve ser no mínimo 1 e no máximo 10.

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
```

```
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O Amazon SES retorna a lista de modelos.

```
node ses_listtemplates.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Obter um modelo de e-mail

Neste exemplo, use um módulo do Node.js para obter um modelo de e-mail a ser usado com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_gettemplate.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar o parâmetro `TemplateName` para o método `GetTemplateCommand` da classe de cliente `SES`. Para chamar o método `GetTemplateCommand`, invoque um objeto de serviço de cliente do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *TEMPLATE_NAME* pelo nome do modelo a ser retornado.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O Amazon SES retorna os detalhes do modelo.

```
node ses_gettemplate.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Criar um modelo de e-mail

Neste exemplo, use um módulo do Node.js para criar um modelo de e-mail a ser usado com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_createtemplate.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar os parâmetros do método `CreateTemplateCommand` da classe de cliente `SES`, incluindo `TemplateName`, `HtmlPart`, `SubjectPart` e `TextPart`. Para chamar o método `CreateTemplateCommand`, invoque um objeto de serviço de cliente do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *TEMPLATE_NAME* por um nome para o novo modelo, *HTML_CONTENT* pelo conteúdo do e-mail marcado com HTML, *SUBJECT* pelo assunto do e-mail e *TEXT_CONTENT* pelo texto do e-mail.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
```

```
    return err;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O modelo é adicionado ao Amazon SES.

```
node ses_createtemplate.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Atualização de um modelo de e-mail

Neste exemplo, use um módulo do Node.js para criar um modelo de e-mail a ser usado com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_updatetemplate.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar os valores do parâmetro `Template` que você deseja atualizar no modelo, com o parâmetro `TemplateName` passado para o método `UpdateTemplateCommand` da classe de cliente `SES`. Para chamar o método `UpdateTemplateCommand`, invoque um objeto de serviço do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode

criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

Note

Substitua *TEMPLATE_NAME* por um nome do modelo, *HTML_CONTENT* pelo conteúdo do e-mail marcado com HTML, *SUBJECT* pelo assunto do e-mail e *TEXT_CONTENT* pelo texto do e-mail.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O Amazon SES retorna os detalhes do modelo.

```
node ses_updatetemplate.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Exclusão de um modelo de e-mail

Neste exemplo, use um módulo do Node.js para criar um modelo de e-mail a ser usado com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_deletetemplate.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar o parâmetro `TemplateName` obrigatório para o método `DeleteTemplateCommand` da classe de cliente `SES`. Para chamar o método `DeleteTemplateCommand`, invoque um objeto de serviço do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *TEMPLATE_NAME* pelo nome do modelo a ser excluído.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O Amazon SES retorna os detalhes do modelo.

```
node ses_deletetemplate.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Envio de e-mail usando o Amazon SES

Este exemplo de código Node.js mostra:

- Envie uma e-mail de texto ou HTML.
- Envie e-mails usando um modelo de e-mail.
- Envie e-mails em massa usando um modelo de e-mail.

A API do Amazon SES fornece duas maneiras diferentes para você enviar um e-mail, dependendo de quanto controle você deseja ter sobre a composição da mensagem de e-mail: formatada e bruta. Para obter detalhes, consulte Enviar e-mail formatado usando a API do Amazon SES e Enviar e-mail bruto usando a API do Amazon SES.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para enviar e-mails de várias maneiras. Os módulos Node.js usam o SDK JavaScript para criar e usar modelos de e-mail usando esses métodos da classe SES cliente:

- `SendEmailCommand`
- `SendTemplatedEmailCommand`
- `SendBulkTemplatedEmailCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses TypeScript exemplos de Node e instale os módulos necessários AWS SDK for JavaScript e de terceiros. Siga as instruções em GitHub.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS .

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.

- Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Requisitos de envio de mensagens de e-mail

O Amazon SES compõe uma mensagem de e-mail e imediatamente a coloca na fila para envio. Para enviar e-mail usando o método `SendEmailCommand`, sua mensagem deve atender aos seguintes requisitos:

- Você deve enviar a mensagem a partir de um domínio ou endereço de e-mail verificado. Se você tentar enviar um e-mail usando um domínio ou endereço não verificados, a operação resultará no erro `"Email address not verified"`.
- Se sua conta ainda estiver na sandbox do Amazon SES, você só poderá enviar para endereços ou domínios verificados ou para endereços de e-mail associados simulador de caixa postal do Amazon SES. Para obter mais informações, consulte Verificar endereços de e-mail e domínios no Guia do desenvolvedor do Amazon Simple Email Service.
- O tamanho total da mensagem, incluindo anexos, deve ser menor que 10 MB.
- A mensagem deve incluir pelo menos um endereço de e-mail de destinatário. O endereço do destinatário pode ser um endereço Para:, um endereço CC: ou um endereço CCO:. Se o endereço de e-mail do destinatário for inválido (ou seja, não estiver no formato `UserName@[SubDomain.]Domain.TopLevelDomain`), a mensagem inteira será rejeitada, mesmo se a mensagem contiver outros destinatários válidos.
- A mensagem não pode incluir mais de 50 destinatários nos campos Para:, CC: e CCO:. Se você precisar enviar uma mensagem de e-mail para um público maior, pode dividir a lista de destinatários em grupos de 50 ou menos e chamar o método `sendEmail` várias vezes para enviar a mensagem para cada grupo.

Enviar um e-mail

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
```

```
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_sendemail.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para transmitir os valores de parâmetro que definem o e-mail a ser enviado, incluindo endereços do remetente e do destinatário, assunto e corpo do e-mail em texto sem formatação e formato HTML, para o método `SendEmailCommand` da classe de cliente `SES`. Para chamar o método `SendEmailCommand`, invoque um objeto de serviço do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *`RECEIVER_ADDRESS`* pelo endereço para o qual enviar o e-mail, e *`SENDER_ADDRESS`* pelo endereço de e-mail do qual enviar o e-mail.

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
```

```
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
```

```
    }
    throw caught;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O e-mail é colocado na fila para ser enviado pelo Amazon SES.

```
node ses_sendemail.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Enviar um e-mail usando um modelo

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_sendtemplatedemail.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para transmitir os valores de parâmetro que definem o e-mail a ser enviado, incluindo endereços do remetente e do destinatário, assunto, corpo do e-mail em texto sem formatação e formato HTML, para o método `SendTemplatedEmailCommand` da classe de cliente `SES`. Para chamar o método `SendTemplatedEmailCommand`, invoque um objeto de serviço de cliente do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *`REGION`* pela sua AWS região, *`RECEIVER_ADDRESS`* pelo endereço para o qual enviar o e-mail, *`SENDER_ADDRESS pelo endereço`* de e-mail do qual enviar o e-mail e *`TEMPLATE_NAME`* pelo nome do modelo.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the party
 gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
```

```
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O e-mail é colocado na fila para ser enviado pelo Amazon SES.

```
node ses_sendtemplatedemail.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Enviar um e-mail em massa usando um modelo

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `sesClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SES. Substitua *`REGION`* pela sua AWS região.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `ses_sendbulktemplatedemail.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para transmitir os valores de parâmetro que definem o e-mail a ser enviado, incluindo endereços do remetente e do destinatário, assunto e corpo do e-mail em texto sem formatação e formato HTML, para o método `SendBulkTemplatedEmailCommand` da classe de cliente `SES`. Para chamar o método `SendBulkTemplatedEmailCommand`, invoque um objeto de serviço do Amazon SES, passando os parâmetros.

> **Note**
>
> Este exemplo importa e usa os clientes do pacote AWS Service V3 necessários, os comandos V3 e usa o `send` método em um padrão async/await. Em vez disso, você pode criar esse exemplo usando comandos da V2 fazendo algumas pequenas alterações. Para obter detalhes, consulte Usando comandos v3.

> **Note**
>
> Substitua *`RECEIVER_ADDRESSES`* pelo endereço para o qual enviar o e-mail, e *`SENDER_ADDRESS`* pelo endereço de e-mail do qual enviar o e-mail.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
```

```
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map each
 user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
```

```
    }
    throw caught;
  }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando. O e-mail é colocado na fila para ser enviado pelo Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Exemplos do Amazon Simple Notification Service

O Amazon Simple Notification Service (Amazon SNS) é um serviço da Web que coordena e gerencia a entrega ou o envio de mensagens para endpoints ou clientes inscritos.

No Amazon SNS, há dois tipos de clientes – os publicadores e os assinantes – também chamados de produtores e consumidores.

Os editores se comunicam de maneira assíncrona com os inscritos produzindo e enviando uma mensagem para um tópico, que é um canal de comunicação e um ponto de acesso lógico. Os assinantes (servidores da Web, endereços de e-mail, filas do Amazon SQS, funções do AWS Lambda) consomem ou recebem a mensagem ou a notificação em um dos protocolos compatíveis (Amazon SQS, HTTP/S, e-mail, SMS, AWS Lambda) quando estão inscritos no tópico.

A API JavaScript para Amazon SNS é exposta por meio da Classe: SNS.

Tópicos

- Gerenciamento de tópicos no Amazon SNS

- Publicação de mensagens no Amazon SNS
- Gerenciamento de assinaturas no Amazon SNS
- Envio de mensagens SMS com o Amazon SNS

Gerenciamento de tópicos no Amazon SNS

Este exemplo de código Node.js mostra:

- Como criar tópicos no Amazon SNS para os quais você pode publicar notificações.
- Como excluir tópicos criados no Amazon SNS.
- Como obter uma lista de tópicos disponíveis.
- Como obter e definir atributos de tópicos.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para criar, listar e excluir tópicos do Amazon SNS e para lidar com atributos de tópicos. Os módulos do Node.js usam o SDK para JavaScript para gerenciar tópicos usando estes métodos da classe de cliente SNS:

- `CreateTopicCommand`
- `ListTopicsCommand`
- `DeleteTopicCommand`
- `GetTopicAttributesCommand`
- `SetTopicAttributesCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Criar um tópico

Neste exemplo, use um módulo do Node.js para criar um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `create-topic.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto para passar o `Name` para o novo tópico para o método `CreateTopicCommand` da classe de cliente `SNS`. Para chamar o método `CreateTopicCommand`, crie uma função assíncrona que invoca um objeto de serviço do Amazon SNS, passando o objeto dos parâmetros. O `data` retornado contém o ARN do tópico.

> **Note**
>
> Substitua *TOPIC_NAME* pelo nome do tópico.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node create-topic.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Listar os tópicos do

Neste exemplo, use um módulo do Node.js para listar todos os tópicos do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `list-topics.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto vazio para passar para o método `ListTopicsCommand` da classe de cliente SNS. Para chamar o método `ListTopicsCommand`, crie uma função assíncrona que invoca um objeto de serviço do Amazon SNS, passando o objeto dos parâmetros. O `data` retornado contém uma matriz dos nomes dos recursos da Amazon (ARNs) do tópico.

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node list-topics.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exclusão de um tópico

Neste exemplo, use um módulo do Node.js para excluir um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `delete-topic.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto contendo o `TopicArn` do tópico para excluir e passar para o método `DeleteTopicCommand` da classe de cliente `SNS`. Para chamar o método `DeleteTopicCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`TOPIC_ARN`* pelo nome do recurso da Amazon (ARN) do tópico que você está excluindo.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
```

```
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node delete-topic.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Obter atributos do tópico

Neste exemplo, use um módulo do Node.js para recuperar atributos de um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `get-topic-attributes.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo o `TopicArn` de um tópico para excluir e passar para o método `GetTopicAttributesCommand` da classe de cliente `SNS`. Para chamar o método `GetTopicAttributesCommand`, invoque um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`TOPIC_ARN`* pelo ARN do tópico.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
```

```
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node get-topic-attributes.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Definir atributos do tópico

Neste exemplo, use um módulo do Node.js para definir os atributos mutáveis de um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `set-topic-attributes.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo os parâmetros para a atualização do atributo, incluindo o `TopicArn` do tópico cujos atributos você deseja definir, o nome do atributo a ser definido e o novo valor desse atributo. É possível definir apenas os atributos `Policy`, `DisplayName` e `DeliveryPolicy`. Passe os parâmetros para o método `SetTopicAttributesCommand` da classe de cliente `SNS`. Para chamar o método `SetTopicAttributesCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *ATTRIBUTE_NAME* pelo nome do atributo que você está definindo, *TOPIC_ARN* pelo nome do recurso da Amazon (ARN) do tópico cujos atributos você deseja definir e *NEW_ATTRIBUTE_VALUE* pelo novo valor desse atributo.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node set-topic-attributes.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Publicação de mensagens no Amazon SNS

Este exemplo de código Node.js mostra:

- Como publicar mensagens em um tópico do Amazon SNS.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para publicar mensagens do Amazon SNS nos endpoints do tópico, e-mails ou números de telefone. Os módulos do Node.js usam o SDK para JavaScript para enviar mensagens usando este método da classe de cliente SNS:

- `PublishCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Publicar uma mensagem em um tópico do SNS

Neste exemplo, use um módulo do Node.js para publicar uma mensagem em um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `publish-topic.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto que contenha os parâmetros para publicar uma mensagem, incluindo o texto da mensagem e o nome do recurso da Amazon (ARN) do tópico do Amazon SNS. Para obter detalhes sobre os atributos de SMS disponíveis, consulte SetSMSAttributes.

Passe os parâmetros para o método `PublishCommand` da classe de cliente `SNS`. Crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`MESSAGE_TEXT`* pelo texto da mensagem e *`TOPIC_ARN`* pelo ARN do tópico do SNS.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
```

```
 *                                                    if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node publish-topic.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Gerenciamento de assinaturas no Amazon SNS

Este exemplo de código Node.js mostra:

- Como listar todas as assinaturas para um tópico do Amazon SNS.
- Como inscrever um endereço de e-mail, um endpoint de aplicativo ou uma função do AWS Lambda em um tópico do Amazon SNS.
- Como cancelar a assinatura dos tópicos do Amazon SNS.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para publicar mensagens de notificação em tópicos do Amazon SNS. Os módulos do Node.js usam o SDK para JavaScript para gerenciar tópicos usando estes métodos da classe de cliente SNS:

- `ListSubscriptionsByTopicCommand`
- `SubscribeCommand`
- `ConfirmSubscriptionCommand`
- `UnsubscribeCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Listar assinaturas em um tópico

Neste exemplo, use um módulo do Node.js para listar todas as assinaturas para um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `list-subscriptions-by-topic.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo o parâmetro `TopicArn` para o tópico cujas assinaturas você deseja listar. Passe os parâmetros para o método `ListSubscriptionsByTopicCommand` da classe de cliente SNS. Para chamar o método `ListSubscriptionsByTopicCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`TOPIC_ARN`* pelo nome do recurso da Amazon (ARN) do tópico cujas assinaturas você deseja listar.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
```

```
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node list-subscriptions-by-topic.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Inscrever um endereço de e-mail em um tópico

Neste exemplo, use um módulo do Node.js para inscrever um endereço de e-mail de forma que ele receba mensagens de e-mail SMTP de um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `subscribe-email.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto que contém o parâmetro `Protocol` para especificar o protocolo `email`, o `TopicArn` do tópico a ser assinado e um endereço de e-mail como `Endpoint` da mensagem. Passe os parâmetros para o método `SubscribeCommand` da classe de cliente SNS. Você pode usar o método `subscribe` para assinar vários endpoints diferentes para um tópico do Amazon SNS, dependendo dos valores usados para os parâmetros passados, como outros exemplos neste tópico mostrarão.

Para chamar o método `SubscribeCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`TOPIC_ARN`* pelo nome do recurso da Amazon (ARN) do tópico e *`EMAIL_ADDRESS`* pelo endereço de e-mail para assinar.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
```

```
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node subscribe-email.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Confirmação de assinaturas

Neste exemplo, use um módulo do Node.js para verificar a intenção do proprietário de um endpoint de receber mensagens validando o token enviado para o endpoint por uma ação de assinatura anterior.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

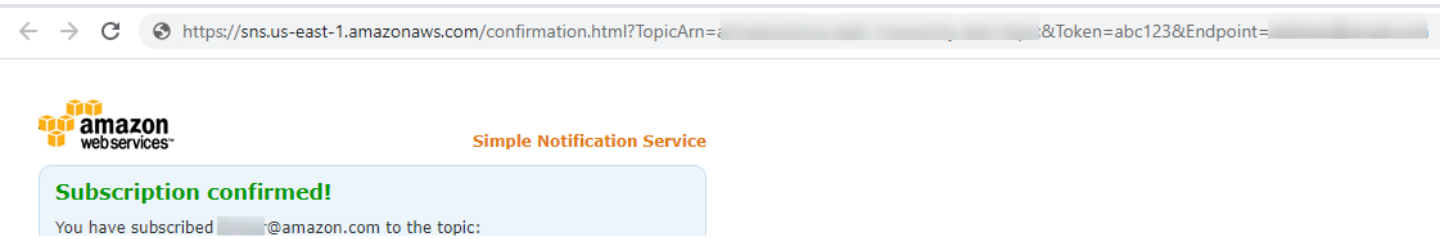
```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `confirm-subscription.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Defina os parâmetros, incluindo `TOPIC_ARN` e `TOKEN`, e defina um valor de `TRUE` ou `FALSE` para `AuthenticateOnUnsubscribe`.

O token é um token de curta duração enviado ao proprietário de um endpoint durante uma ação de `SUBSCRIBE` anterior. Por exemplo, para um endpoint de e-mail, o `TOKEN` está no URL do e-mail de confirmação da assinatura enviado ao proprietário do e-mail. Por exemplo, `abc123` é o token no seguinte URL.



Para chamar o método `ConfirmSubscriptionCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`TOPIC_ARN`* pelo nome do recurso da Amazon (ARN) do tópico, *`TOKEN`* pelo valor do token do URL enviado ao proprietário do endpoint em uma ação de `Subscribe` anterior e defina *`AuthenticateOnUnsubscribe`* com um valor de `TRUE` ou `FALSE`.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
```

```
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node confirm-subscription.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Inscrever um endpoint de aplicativo em um tópico

Neste exemplo, use um módulo do Node.js para inscrever um endpoint de aplicativo móvel para receber notificações de um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `subscribe-app.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos módulos e pacotes necessários.

Crie um objeto contendo o parâmetro `Protocol` para especificar o protocolo `application`, o `TopicArn` para o tópico no qual será feita a assinatura e o nome do recurso da Amazon (ARN) do endpoint de um aplicativo móvel para o parâmetro `Endpoint`. Passe os parâmetros para o método `SubscribeCommand` da classe de cliente SNS.

Para chamar o método `SubscribeCommand`, crie uma função assíncrona que invoca um objeto de serviço do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`TOPIC_ARN`* pelo nome do recurso da Amazon (ARN) do tópico e *`MOBILE_ENDPOINT_ARN`* pelo endpoint que você está assinando o tópico.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 created
 *                            when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
```

```
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node subscribe-app.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Assinatura de uma função do Lambda em um tópico

Neste exemplo, use um módulo do Node.js para inscrever uma função do AWS Lambda para receber notificações de um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `subscribe-lambda.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo o parâmetro `Protocol`, especificando o protocolo `lambda`, o `TopicArn` do tópico de inscrição e o nome do recurso da Amazon (ARN) de uma função AWS Lambda como o parâmetro do `Endpoint`. Passe os parâmetros para o método `SubscribeCommand` da classe de cliente SNS.

Para chamar o método `SubscribeCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`TOPIC_ARN`* pelo nome do recurso da Amazon (ARN) do tópico e *`LAMBDA_FUNCTION_ARN`* pelo nome do recurso da Amazon (ARN) da função do Lambda.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node subscribe-lambda.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Cancelar a inscrição em um tópico

Neste exemplo, use um módulo do Node.js para cancelar a assinatura de um tópico do Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `unsubscribe.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários.

Crie um objeto que contém o parâmetro `SubscriptionArn`, especificando o nome do recurso da Amazon (ARN) da assinatura para cancelar a assinatura. Passe os parâmetros para o método `UnsubscribeCommand` da classe de cliente SNS.

Para chamar o método `UnsubscribeCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`TOPIC_SUBSCRIPTION_ARN`* pelo nome do recurso da Amazon (ARN) da assinatura para cancelar a assinatura.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node unsubscribe.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Envio de mensagens SMS com o Amazon SNS

Este exemplo de código Node.js mostra:

- Como obter e definir as preferências de mensagens SMS para Amazon SNS.
- Como verificar um número de telefone para definir se ele não permite o recebimento de mensagens SMS.
- Como obter uma lista de números de telefone que cancelaram o recebimento de mensagens SMS.
- Como enviar uma mensagem SMS.

O cenário

Você pode usar o Amazon SNS para enviar mensagens de texto ou mensagens SMS para dispositivos habilitados para SMS. Você pode enviar uma mensagem diretamente para um número de telefone, ou enviar uma mensagem para vários números de telefone de uma só vez inscrevendo esses números em um tópico e enviando sua mensagem para o tópico.

Neste exemplo, você usa uma série de módulos do Node.js para publicar mensagens de texto SMS do Amazon SNS em dispositivos habilitados para SMS. Os módulos do Node.js usam o SDK para JavaScript para publicar mensagens SMS usando estes métodos da classe de cliente SNS:

- `GetSMSAttributesCommand`
- `SetSMSAttributesCommand`
- `CheckIfPhoneNumberIsOptedOutCommand`
- `ListPhoneNumbersOptedOutCommand`
- `PublishCommand`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Obter atributos do SMS

Use o Amazon SNS para especificar as preferências para o uso de mensagens SMS, por exemplo, como suas entregas serão otimizadas (para fins de custo ou confiabilidade), o limite de gastos mensais, como as entregas de mensagens serão registradas e a assinatura em relatórios diários de uso de SMS. Essas preferências são recuperadas e definidas como atributos de SMS para Amazon SNS.

Neste exemplo, use um módulo do Node.js para obter os atributos de SMS atuais no Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `get-sms-attributes.js`.

Configure o SDK conforme mostrado anteriormente, incluindo o download dos clientes e pacotes necessários. Crie um objeto contendo os parâmetros para obter atributos de SMS, incluindo os nomes dos atributos individuais a serem obtidos. Para obter detalhes sobre os atributos de SMS disponíveis, consulte SetSMSAttributes na Referência de API do Amazon Simple Notification Service.

Esse exemplo usa o atributo `DefaultSMSType`, que controla se serão enviadas mensagens SMS como `Promotional`, o que otimiza a entrega de mensagens para gerar custos mais baixos, ou como `Transactional`, que otimiza a entrega de mensagens para gerar a mais alta confiabilidade. Passe os parâmetros para o método `SetTopicAttributesCommand` da classe de cliente `SNS`. Para chamar o método `SetSMSAttributesCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *`ATTRIBUTE_NAME`* pelo nome do atributo.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
```

```
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node get-sms-attributes.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Definir atributos do SMS

Neste exemplo, use um módulo do Node.js para obter os atributos de SMS atuais no Amazon SNS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `set-sms-attribute-type.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie um objeto contendo os parâmetros para definir atributos de SMS, incluindo os nomes dos atributos individuais a serem definidos e os valores para cada um. Para obter detalhes sobre os atributos de SMS disponíveis, consulte SetSMSAttributes na Referência de API do Amazon Simple Notification Service.

Este exemplo define o atributo `DefaultSMSType` para `Transactional`, que otimiza a entrega de mensagens para gerar a mais alta confiabilidade. Passe os parâmetros para o método `SetTopicAttributesCommand` da classe de cliente `SNS`. Para chamar o método `SetSMSAttributesCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node set-sms-attribute-type.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Verificar se um número de telefone cancelou o recebimento

Neste exemplo, use um módulo do Node.js para verificar um número de telefone e determinar se ele cancelou o recebimento de mensagens SMS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *REGION* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `check-if-phone-number-is-opted-out.js`. Configure o SDK como mostrado anteriormente. Crie um objeto contendo o número de telefone a ser verificado como parâmetro.

Este exemplo define o parâmetro `PhoneNumber` para especificar o número de telefone a ser verificado. Passe o objeto para o método `CheckIfPhoneNumberIsOptedOutCommand` da classe de cliente `SNS`. Para chamar o método `CheckIfPhoneNumberIsOptedOutCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> 1.
>
> Substitua *PHONE_NUMBER* pelo número de telefone.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
```

```
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node check-if-phone-number-is-opted-out.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Listar números de telefone que cancelaram o recebimento

Neste exemplo, use um módulo do Node.js para obter uma lista de telefones que cancelaram o recebimento de mensagens SMS.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *`REGION`* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `list-phone-numbers-opted-out.js`. Configure o SDK como mostrado anteriormente. Crie um objeto vazio como parâmetro.

Passe o objeto para o método `ListPhoneNumbersOptedOutCommand` da classe de cliente SNS. Para chamar o método `ListPhoneNumbersOptedOutCommand`, crie uma função assíncrona que invoca um objeto de serviço de cliente do Amazon SNS, passando o objeto dos parâmetros.

```
import { ListPhoneNumbersOptedOutCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listPhoneNumbersOptedOut = async () => {
  const response = await snsClient.send(
    new ListPhoneNumbersOptedOutCommand({}),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '44ff72fd-1037-5042-ad96-2fc16601df42',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   phoneNumbers: ['+15555550100']
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node list-phone-numbers-opted-out.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Publicar uma mensagem SMS

Neste exemplo, use um módulo do Node.js para enviar uma mensagem SMS a um número de telefone.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `snsClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon SNS. Substitua *REGION* pela sua região da AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Crie um módulo do Node.js com o nome de arquivo `publish-sms.js`. Configure o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie um objeto contendo os parâmetros `Message` e `PhoneNumber`.

Ao enviar uma mensagem SMS, especifique o número de telefone usando o formato E.164. E.164 é um padrão para a estrutura de número de telefone usada para telecomunicações internacionais. Números de telefone que seguem esse formato podem ter um máximo de 15 dígitos, e eles são prefixados com o caractere de mais (+) e o código do país. Por exemplo, um número de telefone dos EUA no formato E.164 seria exibido como +1001XXX5550100.

Este exemplo define o parâmetro `PhoneNumber` para especificar o número de telefone ao qual a mensagem deve ser enviada. Passe o objeto para o método `PublishCommand` da classe de cliente SNS. Para chamar o método `PublishCommand`, crie uma função assíncrona que invoca um objeto de serviço do Amazon SNS, passando o objeto dos parâmetros.

> **Note**
>
> Substitua *TEXT_MESSAGE* pela mensagem de texto e *PHONE_NUMBER* pelo número de telefone.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
```

```
 *                                                    if you are using the `json`
 `MessageStructure`.
 * @param {*} phoneNumber - The phone number to send the message to.
 */
export const publish = async (
  message = "Hello from SNS!",
  phoneNumber = "+15555555555",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      // One of PhoneNumber, TopicArn, or TargetArn must be specified.
      PhoneNumber: phoneNumber,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '7410094f-efc7-5f52-af03-54737569ab77',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node publish-sms.js
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do Amazon Transcribe

O Amazon Transcribe facilita para os desenvolvedores adicionarem o recurso de conversão de fala em texto aos seus aplicativos.

A JavaScript API do Amazon Transcribe é exposta por meio da TranscribeServiceclasse cliente.

Tópicos

- Exemplos do Amazon Transcribe
- Exemplos do Amazon Transcribe Medical

Exemplos do Amazon Transcribe

Neste exemplo, uma série de módulos do Node.js é usada para criar, listar e excluir trabalhos de transcrição que usam os seguintes métodos da classe de cliente do `TranscribeService`:

- `StartTranscriptionJobCommand`
- `ListTranscriptionJobsCommand`
- `DeleteTranscriptionJobCommand`

Para obter mais informações sobre usuários do Amazon Transcribe, consulte o Guia do desenvolvedor do Amazon Transcribe.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses TypeScript exemplos de Node e instale os módulos necessários AWS SDK for JavaScript e de terceiros. Siga as instruções em GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS .

Important

Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).

- Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
- Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Começar um trabalho do Amazon Transcribe

Este exemplo demonstra como iniciar um trabalho de transcrição do Amazon Transcribe usando o AWS SDK for JavaScript. Para obter mais informações, consulte StartTranscriptionJobCommand.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `transcribeClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Transcribe. Substitua *`REGION`* pela sua AWS região.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `transcribe-create-job.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie um objeto de parâmetros, especificando os parâmetros necessários. Inicie o trabalho usando o comando `StartMedicalTranscriptionJobCommand`.

Note

Substitua *`MEDICAL_JOB_NAME`* por um nome para o trabalho de transcrição. Para *`OUTPUT_BUCKET_NAME`*, especifique o bucket do Amazon S3 em que a saída é salva. Para *`JOB_TYPE`*, especifique os tipos de trabalho. Para *`SOURCE_LOCATION`*, especifique a

localização do arquivo de origem. Para *SOURCE_FILE_LOCATION*, especifique a localização do arquivo de mídia de entrada.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node transcribe-create-job.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Listar trabalhos do Amazon Transcribe

Este exemplo mostra como listar os trabalhos de transcrição do Amazon Transcribe usando o AWS SDK for JavaScript. Para obter mais informações sobre quais outras configurações você pode modificar, consulte `ListTranscriptionJobCommand`.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `transcribeClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Transcribe. Substitua *`REGION`* pela sua AWS região.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `transcribe-list-jobs.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie um objeto de parâmetros com os parâmetros necessários.

> **Note**
>
> Substitua *`KEY_WORD`* por uma palavra-chave que o nome dos trabalhos retornados deve conter.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};
```

```
export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node transcribe-list-jobs.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Exclusão de um trabalho do Amazon Transcribe

Este exemplo mostra como excluir um trabalho de transcrição do Amazon Transcribe usando o AWS SDK for JavaScript. Para obter mais informações sobre comandos opcionais, consulte `DeleteTranscriptionJobCommand`.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `transcribeClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Transcribe. Substitua *`REGION`* pela sua AWS região.

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `transcribe-delete-job.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Especifique a AWS região e o nome do trabalho que você deseja excluir.

> **Note**
>
> Substitua *JOB_NAME* pelo nome do trabalho a ser excluído.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node transcribe-delete-job.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Exemplos do Amazon Transcribe Medical

Neste exemplo, uma série de módulos do Node.js é usada para criar, listar e excluir trabalhos de transcrição médica que usam os seguintes métodos da classe de cliente do `TranscribeService`:

- `StartMedicalTranscriptionJobCommand`
- `ListMedicalTranscriptionJobsCommand`

- `DeleteMedicalTranscriptionJobCommand`

Para obter mais informações sobre usuários do Amazon Transcribe, consulte o Guia do desenvolvedor do Amazon Transcribe.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses TypeScript exemplos de Node e instale os módulos necessários AWS SDK for JavaScript e de terceiros. Siga as instruções em GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS .

> **Important**
>
> Esses exemplos demonstram como importar/exportar objetos e comandos do serviço de cliente usando o ECMAScript6 (ES6).
>
> - Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> - Se você preferir usar a sintaxe do CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Como iniciar um trabalho de transcrição no Amazon Transcribe Medical

Este exemplo demonstra como iniciar um trabalho de transcrição Amazon Transcribe Medical usando o AWS SDK for JavaScript. Para obter mais informações, consulte start MedicalTranscription Job.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `transcribeClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Transcribe. Substitua *`REGION`* pela sua AWS região.

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
```

```
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `transcribe-create-medical-job.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie um objeto de parâmetros, especificando os parâmetros necessários. Inicie o trabalho médico usando o comando `StartMedicalTranscriptionJobCommand`.

> **Note**
>
> Substitua *MEDICAL_JOB_NAME* por um nome para o trabalho de transcrição médica. Para *OUTPUT_BUCKET_NAME*, especifique o bucket do Amazon S3 em que a saída é salva. Para *JOB_TYPE*, especifique os tipos de trabalho. Para *SOURCE_LOCATION*, especifique a localização do arquivo de origem. Para *SOURCE_FILE_LOCATION*, especifique a localização do arquivo de mídia de entrada.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};
```

```
export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node transcribe-create-medical-job.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Como lista trabalhos do Amazon Transcribe Medical

Este exemplo mostra como listar os trabalhos de transcrição do Amazon Transcribe usando o AWS SDK for JavaScript. Para obter mais informações, consulte ListTranscriptionMedicalJobsComando.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `transcribeClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Transcribe. Substitua *`REGION`* pela sua AWS região.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `transcribe-list-medical-jobs.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie um objeto de parâmetros com os parâmetros necessários e liste os trabalhos médicos usando o comando `ListMedicalTranscriptionJobsCommand`.

> **Note**
>
> Substitua *KEYWORD* por uma palavra-chave que o nome dos trabalhos retornados deve conter.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListMedicalTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Returns only transcription job names containing this
 string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListMedicalTranscriptionJobsCommand(params)
    );
    console.log("Success", data.MedicalTranscriptionJobName);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node transcribe-list-medical-jobs.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Exclusão de um trabalho do Amazon Transcribe Medical

Este exemplo mostra como excluir um trabalho de transcrição do Amazon Transcribe usando o AWS SDK for JavaScript. Para obter mais informações sobre comandos opcionais, consulte `DeleteTranscriptionMedicalJobCommand`.

Crie um diretório `libs` e um módulo do Node.js com o nome de arquivo `transcribeClient.js`. Copie e cole o código abaixo nele, o que cria o objeto de cliente do Amazon Transcribe. Substitua *`REGION`* pela sua AWS região.

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Crie um módulo do Node.js com o nome de arquivo `transcribe-delete-job.js`. Certifique-se de configurar o SDK conforme mostrado anteriormente, incluindo a instalação dos clientes e pacotes necessários. Crie um objeto de parâmetros com os parâmetros necessários e exclua o trabalho médico usando o comando `DeleteMedicalJobCommand`.

> **Note**
>
> Substitua *`JOB_NAME`* pelo nome do trabalho a ser excluído.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

Para executar o exemplo, digite o seguinte no prompt de comando.

```
node transcribe-delete-medical-job.js
```

Esse código de exemplo pode ser encontrado aqui em GitHub.

Configuração do Node.js em uma instância do Amazon EC2

Um cenário comum para usar o Node.js com o SDK JavaScript é configurar e executar um aplicativo web Node.js em uma instância do Amazon Elastic Compute Cloud (Amazon EC2). Neste tutorial, você criará uma instância do Linux; conecte-se a ela usando o SSH e instale o Node.js para rodar nessa instância.

Pré-requisitos

Este tutorial pressupõe que você já tenha iniciado uma instância do Linux com um nome DNS público que possa ser acessado na Internet e ao qual você possa se conectar usando o SSH. Para obter mais informações, consulte Etapa 1: iniciar uma instância no Guia do usuário do Amazon EC2.

> **Important**
>
> Use a imagem de máquina da Amazon (AMI) do Amazon Linux 2023 ao iniciar uma nova instância do Amazon EC2.

Você também precisa ter configurado o grupo de segurança para permitir as conexões SSH (porta 22), HTTP (porta 80) e HTTPS (porta 443). Para obter mais informações sobre esses pré-requisitos, consulte Configuração com o Amazon EC2 no Guia do usuário do Amazon EC2.

Procedimento

O procedimento a seguir ajuda você a instalar o Node.js em uma instância do Amazon Linux. Você pode usar esse servidor para hospedar um aplicativo web do Node.js.

Para configurar o Node.js em sua instância do Linux:

1. Conecte-se à sua instância do Linux como `ec2-user` usando SSH.
2. Instale o gerenciador de versão do nó (`nvm`) digitando o seguinte na linha de comando.

   > **Warning**
   >
   > AWS não controla o código a seguir. Antes de executar, certifique-se de verificar sua autenticidade e integridade. Mais informações sobre esse código podem ser encontradas no repositório nvm. GitHub

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
   ```

   Usaremos o `nvm` para instalar o Node.js, pois o `nvm` pode instalar várias versões do Node.js e permitir que você alterne entre elas.
3. Para carregar o `nvm`, digite o seguinte na linha de comando:

   ```
   source ~/.bashrc
   ```
4. Use o nvm para instalar a versão do LTS mais recente do Node.js digitando o seguinte na linha de comando.

   ```
   nvm install --lts
   ```

   Instalar o Node.js também instala o gerenciador de pacotes do nó (`npm`) para que você possa instalar módulos adicionais, conforme necessário.
5. Verifique se o Node.js está instalado e funcionando corretamente ao digitar o seguinte na linha de comando.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Isso exibe a seguinte mensagem que mostra a versão do Node.js em execução.

   ```
   Running Node.js VERSION
   ```

> **Note**
>
> A instalação do nó se aplica somente à sessão atual do Amazon EC2. Se você reiniciar sua sessão de CLI, precisará usar o nvm novamente para habilitar a versão do nó instalada. Se a instância for concluída, você precisará instalar o nó novamente. A alternativa é criar uma imagem de máquina da Amazon (AMI) da instância do Amazon EC2 assim que você tiver a configuração que deseja manter, conforme descrito no tópico a seguir.

Criar uma imagem de máquina da Amazon (AMI)

Depois de instalar o Node.js em uma instância do Amazon EC2, você pode criar uma imagem de máquina da Amazon (AMI) a partir dessa instância. A criação de uma AMI facilita o provisionamento de várias instâncias do Amazon EC2 com a mesma instalação do Node.js. Para obter mais informações sobre a criação de uma AMI a partir de uma instância existente, consulte Criação de uma AMI Linux baseada no Amazon EBS no Guia do usuário do Amazon EC2.

Recursos relacionados

Para obter mais informações sobre os comandos e o software usados neste tópico, consulte as seguintes páginas da Web:

- Gerenciador de versões do Node (`nvm`) — Veja o repositório nvm ativado. GitHub
- gerenciador de pacotes do nó (`npm`): consulte o site do npm.

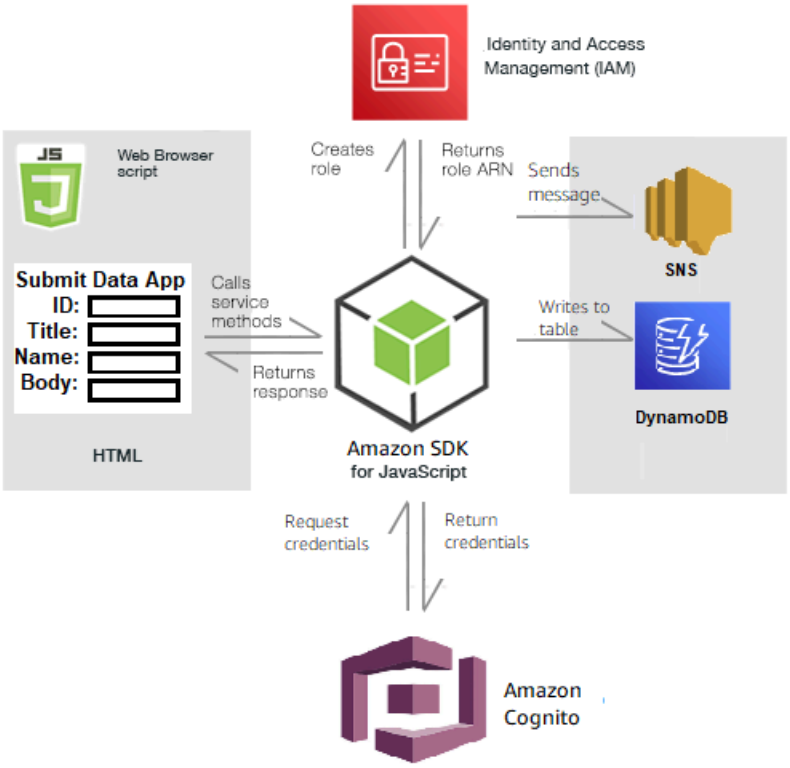
Criar um aplicativo para enviar dados para o DynamoDB

Este tutorial do Node.js entre serviços mostra como criar um aplicativo que permite que os usuários enviem dados para uma tabela do Amazon DynamoDB. Este aplicativo usa os seguintes serviços:

- AWS Identity and Access Management (IAM) e Amazon Cognito para autorização e permissões.
- Amazon DynamoDB (DynamoDB) para criar e atualizar as tabelas.
- Amazon Simple Notification Service (Amazon SNS) para notificar o administrador do aplicativo quando um usuário atualiza a tabela.

O cenário

Neste tutorial, uma página HTML fornece um aplicativo baseado em navegador para enviar dados para uma tabela do Amazon DynamoDB. O aplicativo usa o Amazon SNS para notificar o administrador do aplicativo quando um usuário atualiza a tabela.



Para criar o aplicativo:

1. Pré-requisitos
2. Provisionar recursos
3. Criar o HTML
4. Criar o script do navegador
5. Próximas etapas

Pré-requisitos

Conclua as seguintes tarefas de pré-requisito:

- Configure o ambiente do projeto para executar estes exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

Criar os recursos da AWS

Este aplicativo requer os seguintes recursos:

- Função de usuário não autenticado do Amazon Cognito do AWS Identity and Access Management (IAM) com as seguintes permissões:
  - sns:Publish
  - dynamodb:PutItem
- Uma tabela do DynamoDB.

Você pode criar esses recursos manualmente no console da AWS, mas recomendamos provisioná-los usando o AWS CloudFormation conforme descrito neste tutorial.

Criar os recursos da AWS usando o AWS CloudFormation

O AWS CloudFormation permite que você crie e provisione implantações de infraestrutura da AWS de maneira previsível e repetida. Para mais informações sobre o AWS CloudFormation, consulte o AWS CloudFormationGuia do usuário do .

Para criar a pilha AWS CloudFormation usando a AWS CLI:

1. Instale e configure a AWS CLI seguindo as instruções do Guia do usuário da AWS CLI.
2. Crie um arquivo chamado `setup.yaml` no diretório raiz da pasta do projeto e copie o conteúdo aqui no GitHub para ele.

   > **Note**
   >
   > O modelo AWS CloudFormation foi gerado usando o AWS CDK disponível aqui no GitHub. Para obter mais informações sobre o AWS CDK, consulte o Guia do desenvolvedor do AWS Cloud Development Kit (AWS CDK).

3. Execute o comando a seguir na linha de comando, substituindo *`STACK_NAME`* por um nome exclusivo para a pilha, e *`REGION`* na região da AWS.

> **Important**
>
> O nome da pilha deve ser exclusivo em uma Região da AWS e na conta da AWS. Você pode especificar até 128 caracteres, e números e hifens são permitidos.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM --region REGION
```

Para obter mais informações sobre os parâmetros do comando `create-stack`, consulte o Guia de referência de comandos da AWS CLI e o Guia do usuário do AWS CloudFormation.

Para visualizar os recursos criados, abra o AWS CloudFormation no console de gerenciamento da AWS, escolha a pilha e selecione a guia Recursos.

4. Quando a pilha for criada, use o AWS SDK for JavaScript para preencher a tabela do DynamoDB, conforme descrito em Como preencher a tabela.

Como preencher a tabela

Para preencher a tabela, primeiro crie um diretório chamado `libs`, nele crie um arquivo chamado `dynamoClient.js` e cole o conteúdo abaixo nesse arquivo. Substitua *`REGION`* por sua região da AWS, e substitua *`IDENTITY_POOL_ID`* por um ID do banco de identidades do Amazon Cognito. O objeto de cliente do DynamoDB é criado.

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon DynaomDB service client object.
const dynamoClient = new DynamoDBClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
```

```
  }),
});

export { dynamoClient };
```

Esse código está disponível aqui no GitHub.

Em seguida, crie uma pasta `dynamoAppHelperFiles` na pasta do seu projeto, crie um arquivo `update-table.js` nela e copie o conteúdo aqui no GitHub para ela.

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { dynamoClient } from "../libs/dynamoClient.js";

// Set the parameters
export const params = {
  TableName: "Items",
  Item: {
    id: { N: "1" },
    title: { S: "aTitle" },
    name: { S: "aName" },
    body: { S: "aBody" },
  },
};

export const run = async () => {
  try {
    const data = await dynamoClient.send(new PutItemCommand(params));
    console.log("success");
    console.log(data);
  } catch (err) {
    console.error(err);
  }
};
run();
```

Na linha de comando, execute o comando a seguir.

```
node update-table.js
```

Esse código está disponível aqui no GitHub.

Criar uma página de front-end para o aplicativo

Aqui você cria a página HTML do navegador front-end para o aplicativo.

Crie um diretório `DynamoDBApp`, crie um arquivo chamado `index.html` e copie o código daqui no GitHub. O elemento `script` adiciona o arquivo `main.js`, que contém todo o JavaScript necessário para o exemplo. Você criará o arquivo `main.js` mais adiante neste tutorial. O código restante em `index.html` cria a página do navegador que captura os dados inseridos pelos usuários.

Esse código de exemplo pode ser encontrado aqui no GitHub.

Criar o script do navegador

Primeiro, crie os objetos de cliente de serviço necessários para o exemplo. Crie um diretório `libs`, crie `snsClient.js` e cole o código abaixo nele. Substitua *`REGION`* e *`IDENTITY_POOL_ID`* em cada um.

> **Note**
>
> Use o ID do banco de identidades do Amazon Cognito que você criou em Criar os recursos da AWS.

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { SNSClient } from "@aws-sdk/client-sns";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const snsClient = new SNSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { snsClient };
```

Esse código está disponível aqui no GitHub.

Para criar o script do navegador para este exemplo, em uma pasta chamada `DynamoDBApp`, crie um módulo Node.js com o nome do arquivo `add_data.js` e cole o código abaixo nele. A função `submitData` envia dados para uma tabela do DynamoDB e envia uma mensagem SMS para o administrador do aplicativo usando o Amazon SNS.

Na função `submitData`, declare variáveis para o número de telefone de destino, os valores inseridos na interface do aplicativo e para o nome do bucket do Amazon S3. Em seguida, crie um objeto de parâmetros para adicionar um item à tabela. Se nenhum dos valores estiver vazio, `submitData` adiciona o item à tabela e envia a mensagem. Lembre-se de tornar a função disponível para o navegador, com `window.submitData = submitData`.

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";
import { dynamoClient } from "../libs/dynamoClient.js";

export const submitData = async () => {
  //Set the parameters
  // Capture the values entered in each field in the browser (by id).
  const id = document.getElementById("id").value;
  const title = document.getElementById("title").value;
  const name = document.getElementById("name").value;
  const body = document.getElementById("body").value;
  //Set the table name.
  const tableName = "Items";

  //Set the parameters for the table
  const params = {
    TableName: tableName,
    // Define the attributes and values of the item to be added. Adding ' + "" '
 converts a value to
    // a string.
    Item: {
      id: { N: id + "" },
      title: { S: title + "" },
      name: { S: name + "" },
      body: { S: body + "" },
    },
  };
```

```
  // Check that all the fields are completed.
  if (id != "" && title != "" && name != "" && body != "") {
    try {
      //Upload the item to the table
      await dynamoClient.send(new PutItemCommand(params));
      alert("Data added to table.");
      try {
        // Create the message parameters object.
        const messageParams = {
          Message: "A new item with ID value was added to the DynamoDB",
          PhoneNumber: "PHONE_NUMBER", //PHONE_NUMBER, in the E.164 phone number
 structure.
          // For example, ak standard local formatted number, such as (415) 555-2671,
 is +14155552671 in E.164
          // format, where '1' in the country code.
        };
        // Send the SNS message
        const data = await snsClient.send(new PublishCommand(messageParams));
        console.log(
          "Success, message published. MessageID is " + data.MessageId,
        );
      } catch (err) {
        // Display error message if error is not sent
        console.error(err, err.stack);
      }
    } catch (err) {
      // Display error message if item is no added to table
      console.error(
        "An error occurred. Check the console for further information",
        err,
      );
    }
    // Display alert if all field are not completed.
  } else {
    alert("Enter data in each field.");
  }
};
// Expose the function to the browser
window.submitData = submitData;
```

Esse código de exemplo pode ser encontrado aqui no GitHub.

Por fim, execute o seguinte na linha de comando para empacotar o JavaScript deste exemplo em um arquivo chamado `main.js`:

```
webpack add_data.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> Para obter informações sobre como instalar o webpack, consulte Agrupe aplicativos com o webpack.

Para executar o aplicativo, abra `index.html` no navegador.

Excluir os recursos

Conforme informado no início deste tutorial, certifique-se de encerrar todos os recursos que criar enquanto percorre este tutorial para garantir que você não seja cobrado. Você pode fazer isso excluindo a pilha AWS CloudFormation que criou no tópico Criar os recursos da AWS deste tutorial, da seguinte forma:

1. Abra o AWS CloudFormation no console de gerenciamento da AWS.
2. Abra a página Pilhas e selecione a pilha.
3. Escolha Delete (Excluir).

Para obter mais exemplos entre serviços da AWS, consulte Exemplos entre serviços da AWS SDK for JavaScript.

Criar um aplicativo de transcrição com usuários autenticados

Neste tutorial, você aprenderá a:

- Implementar a autenticação usando um banco de identidades do Amazon Cognito para aceitar usuários federados com um grupo de usuários do Amazon Cognito.
- Usar o Amazon Transcribe para transcrever e exibir gravações de voz no navegador.

O cenário

O aplicativo permite que os usuários se inscrevam com um e-mail e nome de usuário exclusivos. Após a confirmação do e-mail, eles podem gravar mensagens de voz que são automaticamente transcritas e exibidas no aplicativo.

Como funciona

O aplicativo usa dois buckets do Amazon S3, um para hospedar o código do aplicativo e outro para armazenar transcrições. A aplicação usa um grupo de usuários do Amazon Cognito para autenticar seus usuários. Os usuários autenticados têm permissões do IAM para acessar os serviços da AWS necessários.

Na primeira vez que um usuário grava uma mensagem de voz, o Amazon S3 cria uma pasta exclusiva com o nome do usuário no bucket do Amazon S3 para armazenar transcrições. O Amazon Transcribe transcreve a mensagem de voz em texto e a salva em JSON na pasta do usuário. Quando o usuário atualiza o aplicativo, suas transcrições são exibidas e estão disponíveis para download ou exclusão.

O tutorial levará aproximadamente 30 minutos para ser concluído.

Etapas

Para criar o aplicativo:

1. Pré-requisitos
2. Criar os recursos da AWS
3. Criar o HTML
4. Preparar o script do navegador
5. Executar o aplicativo
6. Excluir os recursos

Pré-requisitos

- Configure o ambiente do projeto para executar estes exemplos do Node JavaScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Este exemplo usa ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Criar os recursos da AWS

Esta seção descreve como provisionar recursos da AWS para essa aplicação usando o AWS Cloud Development Kit (AWS CDK).

> **Note**
>
> O AWS CDK é uma estrutura de desenvolvimento de software que permite definir recursos de aplicativos de nuvem. Para obter mais informações, consulte o Guia do desenvolvedor do AWS Cloud Development Kit (AWS CDK).

Para criar recursos para o aplicativo, use o modelo aqui no GitHub para criar uma pilha do AWS CDK usando o Console de gerenciamento do AWS Web Services ou a AWS CLI. Para obter instruções sobre como modificar a pilha ou excluir a pilha e seus recursos associados quando concluir o tutorial, consulte aqui no GitHub.

> **Note**
>
> O nome da pilha deve ser exclusivo em uma Região da AWS e na conta da AWS. Você pode especificar até 128 caracteres. São permitidos números e hifens.

A pilha resultante provisiona automaticamente os recursos a seguir.

- Um banco de identidades do Amazon Cognito com uma função de usuário autenticada.

- Uma política do IAM com permissões para o Amazon S3 e o Amazon Transcribe é anexada à função de usuário autenticado.
- Um grupo de usuários do Amazon Cognito que permite que os usuários se inscrevam e façam login no aplicativo.
- Um bucket do Amazon S3 para hospedar os arquivos de aplicativo.
- Um bucket do Amazon S3 para armazenar as transcrições.

> **Important**
>
> Esse bucket do Amazon S3 permite acesso público READ (LIST), o que permite que qualquer pessoa liste os objetos no bucket e potencialmente use as informações de forma incorreta. Se você não excluir esse bucket do Amazon S3 imediatamente após concluir o tutorial, será altamente recomendável que siga as Práticas recomendadas de segurança do Amazon S3 no Guia do usuário do Amazon Simple Storage Service.

Criar o HTML

Crie um arquivo `index.html` e copie e cole o código abaixo nele. A página apresenta um painel de botões para gravação de mensagens de voz e uma tabela que exibe as mensagens transcritas anteriormente do usuário atual. A tag de script no final do elemento `body` invoca o arquivo `main.js`, que contém todo o script do navegador para o aplicativo. Você cria o main.js usando o Webpack, conforme descrito na seção a seguir deste tutorial.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>title</title>
    <link rel="stylesheet" type="text/css" href="recorder.css">
    <style>
        table, td {
            border: 1px solid black;
        }
    </style>
</head>
<body>
<h2>Record</h2>
<p>
```

```
    <button id="record" onclick="startRecord()"></button>
    <button id="stopRecord" disabled onclick="stopRecord()">Stop</button>
<p id="demo" style="visibility: hidden;"></p>
</p>
<p>
    <audio id="recordedAudio"></audio>
</p>

<h2>My transcriptions</h2>
<table id="myTable1" style ="width:678px;">
</table>
<table id="myTable" style ="width:678px;">
        <tr>
    <td style = "font-weight:bold">Time created</td>
    <td style = "font-weight:bold">Transcription</td>
    <td style = "font-weight:bold">Download</td>
    <td style = "font-weight:bold">Delete</td>
       </tr>
</table>

<script type="text/javascript" src="./main.js"></script>
</body>

</html>
```

Esse exemplo de código está disponível aqui no GitHub.

Preparar o script do navegador

Existem três arquivos, `index.html`, `recorder.js` e `helper.js`, que você deve empacotar em um único `main.js` usando o Webpack. Esta seção descreve em detalhes somente as funções no `index.js` que usam o SDK para JavaScript, que está disponível aqui no GitHub.

> **Note**
>
> `recorder.js` e `helper.js` são obrigatórios, mas, como não contêm o código Node.js, são explicados nos comentários em linha aqui e aqui, respectivamente, no GitHub.

Primeiro, defina os parâmetros. `COGNITO_ID` é o endpoint do grupo de usuários do Amazon Cognito que você criou no tópico Criar os recursos da AWS deste tutorial. O formato é `cognito-idp.`*`AWS_REGION`*`.amazonaws.com/`*`USER_POOL_ID`*. O ID do grupo de usuários é *`ID_TOKEN`*

no token de credenciais da AWS, que é removido do URL do aplicativo pela função `getToken` no arquivo 'helper.js'. Esse token é passado para a variável `loginData`, que fornece os objetos do cliente Amazon Transcribe e Amazon S3 com logins. Substitua *"REGION"* pela Região da AWS e *"BUCKET"* pela Substitua *"IDENTITY_POOL_ID"* por `IdentityPoolId` na página de exemplo do banco de identidades do Amazon Cognito que você criou para este exemplo. Isso também é passado para cada objeto do cliente.

```
// Import the required AWS SDK clients and commands for Node.js
import "./helper.js";
import "./recorder.js";
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import {
  CognitoIdentityProviderClient,
  GetUserCommand,
} from "@aws-sdk/client-cognito-identity-provider";
import { S3RequestPresigner } from "@aws-sdk/s3-request-presigner";
import { createRequest } from "@aws-sdk/util-create-request";
import { formatUrl } from "@aws-sdk/util-format-url";
import {
  TranscribeClient,
  StartTranscriptionJobCommand,
} from "@aws-sdk/client-transcribe";
import {
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
  ListObjectsCommand,
  DeleteObjectCommand,
} from "@aws-sdk/client-s3";
import fetch from "node-fetch";

// Set the parameters.
// 'COGINTO_ID' has the format 'cognito-idp.eu-west-1.amazonaws.com/COGNITO_ID'.
let COGNITO_ID = "COGNITO_ID";
// Get the Amazon Cognito ID token for the user. 'getToken()' is in 'helper.js'.
let idToken = getToken();
let loginData = {
  [COGNITO_ID]: idToken,
};

const params = {
```

```
  Bucket: "BUCKET", // The Amazon Simple Storage Solution (S3) bucket to store the
 transcriptions.
  Region: "REGION", // The AWS Region
  identityPoolID: "IDENTITY_POOL_ID", // Amazon Cognito Identity Pool ID.
};

// Create an Amazon Transcribe service client object.
const client = new TranscribeClient({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});

// Create an Amazon S3 client object.
const s3Client = new S3Client({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});
```

Quando a página HTML é carregada, `updateUserInterface` cria uma pasta com o nome do usuário no bucket do Amazon S3 se for a primeira vez que ele faz login no aplicativo. Caso contrário, ele atualiza a interface do usuário com todas as transcrições das sessões anteriores do usuário.

```
window.onload = async () => {
  // Set the parameters.
  const userParams = {
    // Get the access token. 'GetAccessToken()' is in 'helper.js'.
    AccessToken: getAccessToken(),
  };
  // Create a CognitoIdentityProviderClient client object.
  const client = new CognitoIdentityProviderClient({ region: params.Region });
  try {
    const data = await client.send(new GetUserCommand(userParams));
    const username = data.Username;
    // Export username for use in 'recorder.js'.
```

```
    exports.username = username;
    try {
      // If this is user's first sign-in, create a folder with user's name in Amazon S3
bucket.
      // Otherwise, no effect.
      const Key = `${username}/`;
      try {
        const data = await s3Client.send(
          new PutObjectCommand({ Key: Key, Bucket: params.Bucket })
        );
        console.log("Folder created for user ", data.Username);
      } catch (err) {
        console.log("Error", err);
      }
      try {
        // Get a list of the objects in the Amazon S3 bucket.
        const data = await s3Client.send(
          new ListObjectsCommand({ Bucket: params.Bucket, Prefix: username })
        );
        // Create a variable for the list of objects in the Amazon S3 bucket.
        const output = data.Contents;
        // Loop through the objects, populating a row on the user interface for each
object.
        for (var i = 0; i < output.length; i++) {
          var obj = output[i];
          const objectParams = {
            Bucket: params.Bucket,
            Key: obj.Key,
          };
          // Get the name of the object from the Amazon S3 bucket.
          const data = await s3Client.send(new GetObjectCommand(objectParams));
          // Extract the body contents, a readable stream, from the returned data.
          const result = data.Body;
          // Create a variable for the string version of the readable stream.
          let stringResult = "";
          // Use 'yieldUnit8Chunks' to convert the readable streams into JSON.
          for await (let chunk of yieldUint8Chunks(result)) {
            stringResult += String.fromCharCode.apply(null, chunk);
          }
          // The setTimeout function waits while readable stream is converted into
JSON.
          setTimeout(function () {
            // Parse JSON into human readable transcript, which will be displayed on
user interface (UI).
```

```
            const outputJSON =
              JSON.parse(stringResult).results.transcripts[0].transcript;
            // Create name for transcript, which will be displayed.
            const outputJSONTime = JSON.parse(stringResult)
              .jobName.split("/")[0]
              .replace("-job", "");
            i++;
            //
            // Display the details for the transcription on the UI.
            // 'displayTranscriptionDetails()' is in 'helper.js'.
            displayTranscriptionDetails(
              i,
              outputJSONTime,
              objectParams.Key,
              outputJSON
            );
          }, 1000);
        }
      } catch (err) {
        console.log("Error", err);
      }
    } catch (err) {
      console.log("Error creating presigned URL", err);
    }
  } catch (err) {
    console.log("Error", err);
  }
};

// Convert readable streams.
async function* yieldUint8Chunks(data) {
  const reader = data.getReader();
  try {
    while (true) {
      const { done, value } = await reader.read();
      if (done) return;
      yield value;
    }
  } finally {
    reader.releaseLock();
  }
}
```

Quando o usuário grava uma mensagem de voz para transcrições, `upload` carrega as gravações no bucket do Amazon S3. Essa função é chamada a partir do arquivo `recorder.js`.

```
// Upload recordings to Amazon S3 bucket
window.upload = async function (blob, userName) {
  // Set the parameters for the recording recording.
  const Key = `${userName}/test-object-${Math.ceil(Math.random() * 10 ** 10)}`;
  let signedUrl;

  // Create a presigned URL to upload the transcription to the Amazon S3 bucket when it
 is ready.
  try {
    // Create an Amazon S3RequestPresigner object.
    const signer = new S3RequestPresigner({ ...s3Client.config });
    // Create the request.
    const request = await createRequest(
      s3Client,
      new PutObjectCommand({ Key, Bucket: params.Bucket })
    );
    // Define the duration until expiration of the presigned URL.
    const expiration = new Date(Date.now() + 60 * 60 * 1000);
    // Create and format the presigned URL.
    signedUrl = formatUrl(await signer.presign(request, expiration));
    console.log(`\nPutting "${Key}"`);
  } catch (err) {
    console.log("Error creating presigned URL", err);
  }
  try {
    // Upload the object to the Amazon S3 bucket using a presigned URL.
    response = await fetch(signedUrl, {
      method: "PUT",
      headers: {
        "content-type": "application/octet-stream",
      },
      body: blob,
    });
    // Create the transcription job name. In this case, it's the current date and time.
    const today = new Date();
    const date =
      today.getFullYear() +
      "-" +
      (today.getMonth() + 1) +
```

```
      "-" +
      today.getDate();
    const time =
      today.getHours() + "-" + today.getMinutes() + "-" + today.getSeconds();
    const jobName = date + "-time-" + time;

    // Call the "createTranscriptionJob()" function.
    createTranscriptionJob(
      "s3://" + params.Bucket + "/" + Key,
      jobName,
      params.Bucket,
      Key
    );
  } catch (err) {
    console.log("Error uploading object", err);
  }
};

// Create the AWS Transcribe transcription job.
const createTranscriptionJob = async (recording, jobName, bucket, key) => {
  // Set the parameters for transcriptions job
  const params = {
    TranscriptionJobName: jobName + "-job",
    LanguageCode: "en-US", // For example, 'en-US',
    OutputBucketName: bucket,
    OutputKey: key,
    Media: {
      MediaFileUri: recording, // For example, "https://transcribe-demo.s3-
REGION.amazonaws.com/hello_world.wav"
    },
  };
  try {
    // Start the transcription job.
    const data = await client.send(new StartTranscriptionJobCommand(params));
    console.log("Success - transcription submitted", data);
  } catch (err) {
    console.log("Error", err);
  }
};
```

`deleteTranscription` exclui uma transcrição da interface do usuário e `deleteRow` exclui uma transcrição existente do bucket do Amazon S3. Ambos são acionados pelo botão Excluir na interface do usuário.

```
// Delete a transcription from the Amazon S3 bucket.
window.deleteJSON = async (jsonFileName) => {
  try {
    await s3Client.send(
      new DeleteObjectCommand({
        Bucket: params.Bucket,
        Key: jsonFileName,
      })
    );
    console.log("Success - JSON deleted");
  } catch (err) {
    console.log("Error", err);
  }
};
// Delete a row from the user interface.
window.deleteRow = function (rowid) {
  const row = document.getElementById(rowid);
  row.parentNode.removeChild(row);
};
```

Por fim, execute o seguinte no prompt de comando para empacotar o JavaScript deste exemplo em um arquivo chamado `main.js`:

```
webpack index.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> Para obter informações sobre como instalar o webpack, consulte Agrupe aplicativos com o webpack.

Executar o aplicativo

Você pode visualizar o aplicativo no local abaixo.

```
DOMAIN/login?
client_id=APP_CLIENT_ID&response_type=token&scope=aws.cognito.signin.user.admin+email
+openid+phone+profile&redirect_uri=REDIRECT_URL
```

O Amazon Cognito facilita a execução do aplicativo fornecendo um link no Console de gerenciamento do AWS Web Services. Basta navegar até a configuração do cliente do aplicativo do seu grupo de usuários do Amazon Cognito e selecionar Iniciar IU hospedada. O URL do aplicativo tem o seguinte formato:

> **Important**
>
> A IU hospedada usa como padrão um tipo de resposta de 'código'. No entanto, este tutorial foi desenvolvido para o tipo de resposta 'token', então você precisa alterá-lo.

Excluir os recursos da AWS

Ao concluir o tutorial, você deve excluir os recursos para não incorrer em cobranças desnecessárias. Como você adicionou conteúdo aos dois buckets do Amazon S3, você deve excluí-los manualmente. Em seguida, você pode excluir os recursos restantes usando o Console de gerenciamento do AWS Web Services ou a AWS CLI. Para obter instruções sobre como modificar a pilha ou excluir a pilha e seus recursos associados quando concluir o tutorial, consulte aqui no GitHub.

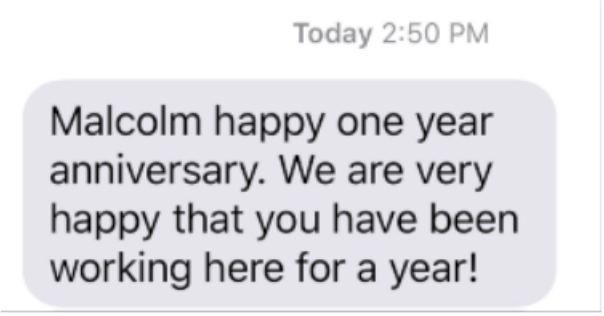
Invocar o Lambda com o API Gateway

Você pode invocar uma função do Lambda usando o Amazon API Gateway, que é um serviço da AWS para criar, publicar, manter, monitorar e proteger APIs de WebSocket, REST e HTTP em qualquer escala. Os desenvolvedores de API podem criar APIs que acessem a AWS ou outros serviços da Web, bem como dados armazenados na Nuvem da AWS. Como um desenvolvedor do API Gateway, você pode criar APIs para usar em suas próprias aplicações cliente. Para obter mais informações, consulte O que é o Amazon API Gateway.

O AWS Lambda é um serviço de computação que permite executar código sem provisionar ou gerenciar servidores. Você pode criar funções do Lambda em várias linguagens de programação. Para obter mais informações sobre o AWS Lambda, consulte O que é o AWS Lambda.

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.

Por exemplo, suponha que uma organização envie uma mensagem de SMS para seus funcionários parabenizando-os pela data de um ano de tempo de casa, conforme mostrado nesta ilustração.



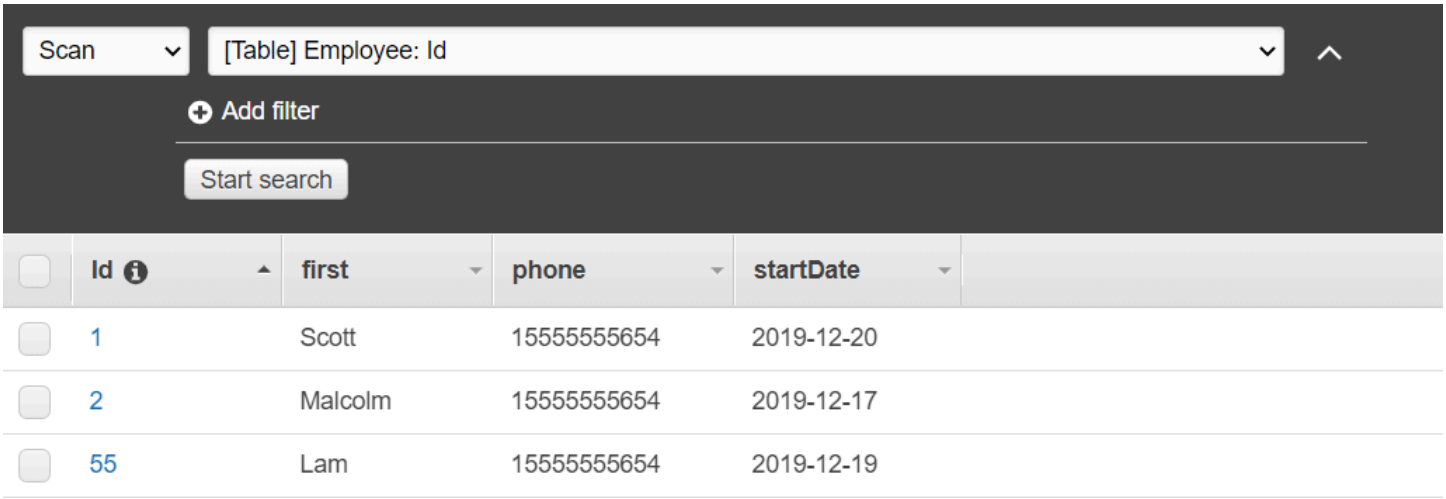
O exemplo levará aproximadamente 20 minutos para ser concluído.

Este exemplo mostra como usar a lógica do JavaScript para criar uma solução que execute este caso de uso. Por exemplo, você aprenderá como ler um banco de dados para determinar quais funcionários atingiram a data de aniversário de um ano, como processar os dados e enviar uma mensagem de texto usando uma função do Lambda. Em seguida, você aprenderá como usar o API Gateway para invocar essa função AWS Lambda usando um endpoint Rest. Por exemplo, você pode invocar a função do Lambda usando este comando curl:

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/
employee"
```

Este tutorial da AWS usa uma tabela do Amazon DynamoDB chamada Funcionários que contém estes campos.

- id - a chave primária da tabela.
- firstName - o nome do funcionário.
- phone - o número de telefone do funcionário.
- startDate - a data de início do funcionário.



> **Important**
>
> Custo para concluir: os serviços da AWS incluídos neste documento estão incluídos no Nível gratuito da AWS. No entanto, certifique-se de encerrar todos os recursos depois de concluir este exemplo para garantir que você não seja cobrado.

Para criar o aplicativo:

1. Concluir os pré-requisitos
2. Criar os recursos da AWS
3. Prepare o script do navegador
4. Criar e carregar a função do Lambda
5. Implantar a função do Lambda
6. Executar o aplicativo
7. Excluir os recursos

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

Criar os recursos da AWS

Este tutorial requer os seguintes recursos:

- Uma tabela do Amazon DynamoDB chamada `Employee` com uma chave chamada `Id` e os campos mostrados na ilustração anterior. Certifique-se de inserir os dados corretos, incluindo um telefone celular válido com o qual você deseja testar esse caso de uso. Para obter mais informações, consulte Criar uma tabela.
- Um perfil do IAM com permissões anexadas para executar funções do Lambda.
- Um bucket do Amazon S3 para hospedar a função do Lambda.

Você pode criar esses recursos manualmente, mas recomendamos provisioná-los usando o AWS CloudFormation conforme descrito neste tutorial.

Criar os recursos da AWS usando o AWS CloudFormation

O AWS CloudFormation permite que você crie e provisione implantações de infraestrutura da AWS de maneira previsível e repetida. Para mais informações sobre o AWS CloudFormation, consulte o AWS CloudFormationGuia do usuário do .

Para criar a pilha AWS CloudFormation usando a AWS CLI:

1. Instale e configure a AWS CLI seguindo as instruções do Guia do usuário da AWS CLI.
2. Crie um arquivo chamado `setup.yaml` no diretório raiz da pasta do projeto e copie o conteúdo aqui no GitHub para ele.

   > **Note**
   >
   > O modelo AWS CloudFormation foi gerado usando o AWS CDK disponível aqui no GitHub. Para obter mais informações sobre o AWS CDK, consulte o Guia do desenvolvedor do AWS Cloud Development Kit (AWS CDK).

3. Execute o comando a seguir na linha de comando, substituindo *`STACK_NAME`* por um nome exclusivo para a pilha.

> **Important**
>
> O nome da pilha deve ser exclusivo em uma Região da AWS e em uma conta da AWS. Você pode especificar até 128 caracteres, e números e hifens são permitidos.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

Para obter mais informações sobre os parâmetros do comando `create-stack`, consulte o Guia de referência de comandos da AWS CLI e o Guia do usuário do AWS CloudFormation.

4. Em seguida, preencha a tabela seguindo o procedimento Como preencher a tabela.

Como preencher a tabela

Para preencher a tabela, primeiro crie um diretório chamado `libs`, nele crie um arquivo chamado `dynamoClient.js` e cole o conteúdo abaixo nesse arquivo.

```
const { DynamoDBClient } = require ( "@aws-sdk/client-dynamodb" );
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
 // Create an Amazon Lambda service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

Esse código está disponível aqui no GitHub.

Em seguida, crie um arquivo chamado `populate-table.js` no diretório raiz da pasta do projeto e copie o conteúdo aqui no GitHub para ele. Para um dos itens, substitua o valor da propriedade `phone` por um número de celular válido no formato E.164, e o valor de `startDate` pela data de hoje.

Na linha de comando, execute o seguinte comando:

```
node populate-table.js
```

```
const { BatchWriteItemCommand } = require ( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Set the parameters.
export const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
```

```
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Esse código está disponível aqui no GitHub.

Criar a função do AWS Lambda

Como configurar o SDK

No diretório `libs`, crie arquivos chamados `snsClient.js` e `lambdaClient.js` e cole o conteúdo abaixo nesses arquivos, respectivamente.

```
const  { SNSClient } = require ( "@aws-sdk/client-sns" );
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon SNS service client object.
const snsClient = new SNSClient({ region: REGION });
module.exports = { snsClient };
```

Substitua *`REGION`* pela Região da AWS. Esse código está disponível aqui no GitHub.

```
const { LambdaClient } = require ( "@aws-sdk/client-lambda" );
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Lambda service client object.
const lambdaClient = new LambdaClient({ region: REGION });
module.exports = { lambdaClient };
```

Substitua *`REGION`* pela Região da AWS. Esse código está disponível aqui no GitHub.

Primeiro, importe os módulos e comandos do AWS SDK for JavaScript (v3) necessários. Em seguida, calcule a data de hoje e atribua-a a um parâmetro. Em terceiro lugar, crie os parâmetros para `ScanCommand`. Substitua *`TABLE_NAME`* pelo nome da tabela que você criou na seção Criar os recursos da AWS deste exemplo.

O snippet de código a seguir mostra essa etapa. (Consulte Como empacotar a função do Lambda para ver o exemplo completo.)

```
"use strict";
const { ScanCommand } = require("@aws-sdk/client-dynamodb");
const { PublishCommand } = require("@aws-sdk/client-sns");
const {snsClient} = require ( "./libs/snsClient" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "Employees",
};
```

Como escanear a tabela do DynamoDB

Primeiro, crie uma função async/await chamada `sendText` para publicar uma mensagem de texto usando o Amazon SNS `PublishCommand`. Em seguida, adicione um padrão de bloco `try` que verifique a tabela do DynamoDB em busca de funcionários com aniversários de trabalho na data de hoje e, em seguida, chame a função `sendText` para enviar uma mensagem de texto a esses funcionários. Se ocorrer um erro, o bloco `catch` será chamado.

O snippet de código a seguir mostra essa etapa. (Consulte Como empacotar a função do Lambda para ver o exemplo completo.)

```
// Helper function to send message using Amazon SNS.
```

```
exports.handler = async () => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      await snsClient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to identify employees with work anniversary today.
    const data = await dynamoClient.send(new ScanCommand(params));
    data.Items.forEach(function (element) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
            "Hi " +
            element.firstName.S +
            "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Como empacotar a função do Lambda

Este tópico descreve como empacotar a `mylambdafunction.ts` e os módulos do AWS SDK for JavaScript necessários para este exemplo em um arquivo empacotado chamado `index.js`.

1. Caso ainda não tenha feito isso, siga as instruções descritas em Tarefas de pré-requisito para este exemplo para instalar o webpack.

> **Note**
>
> Para obter informações sobre o webpack, consulte Agrupe aplicativos com o webpack.

2. Execute o seguinte na linha de comando para empacotar o JavaScript deste exemplo em um arquivo chamado <index.js>:

```
webpack mylambdafunction.ts --mode development --target node --devtool false --
output-library-target umd -o index.js
```

> **Important**
>
> A saída é chamada `index.js`. Isso ocorre porque as funções do Lambda precisam ter um manipulador `index.js` para funcionar.

3. Comprima o arquivo de saída empacotado, `index.js`, em um arquivo ZIP chamado `mylambdafunction.zip`.
4. Faça o upload de `mylambdafunction.zip` para o bucket Amazon S3 que você criou no tópico no Criar os recursos da AWS deste tutorial.

Implantar a função do Lambda

Na raiz do projeto, crie um arquivo `lambda-function-setup.ts` e cole o conteúdo abaixo nele.

Substitua *BUCKET_NAME* pelo nome do bucket Amazon S3 para o qual você fez o upload da versão ZIP da sua função do Lambda. Substitua *ZIP_FILE_NAME* pelo nome da versão ZIP da sua função do Lambda. Substitua *ROLE* pelo Número de recurso da Amazon (ARN) do perfil do IAM que você criou no tópico Criar os recursos da AWS deste tutorial. Substitua *LAMBDA_FUNCTION_NAME* por um nome para a função do Lambda.

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand
} = require ( "@aws-sdk/client-lambda" );
const { lambdaClient} = require ( "./libs/lambdaClient.js );

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
```

```
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
 Services (Amazon SNS) to " +
    "send employees an email on each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambdaClient.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

Insira o seguinte na linha de comando para implantar a função do Lambda.

```
node lambda-function-setup.ts
```

Esse exemplo de código está disponível aqui no GitHub.

Configurar o API Gateway para invocar a função do Lambda

Para criar o aplicativo:

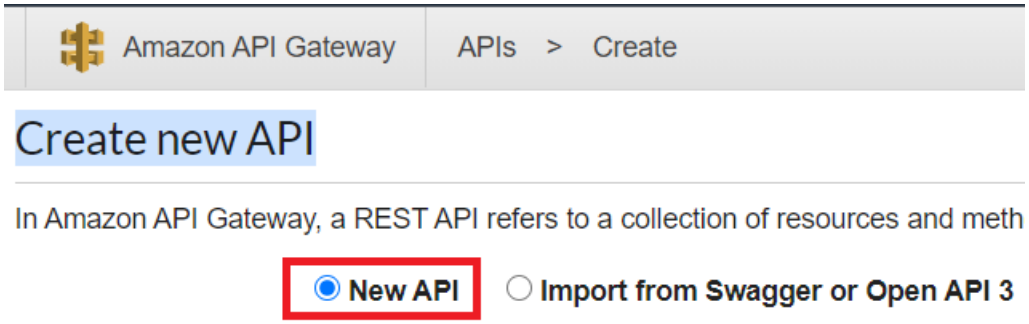
1. Criar a API REST
2. Testar o método do API Gateway
3. Implantar o método do API Gateway

Criar a API REST

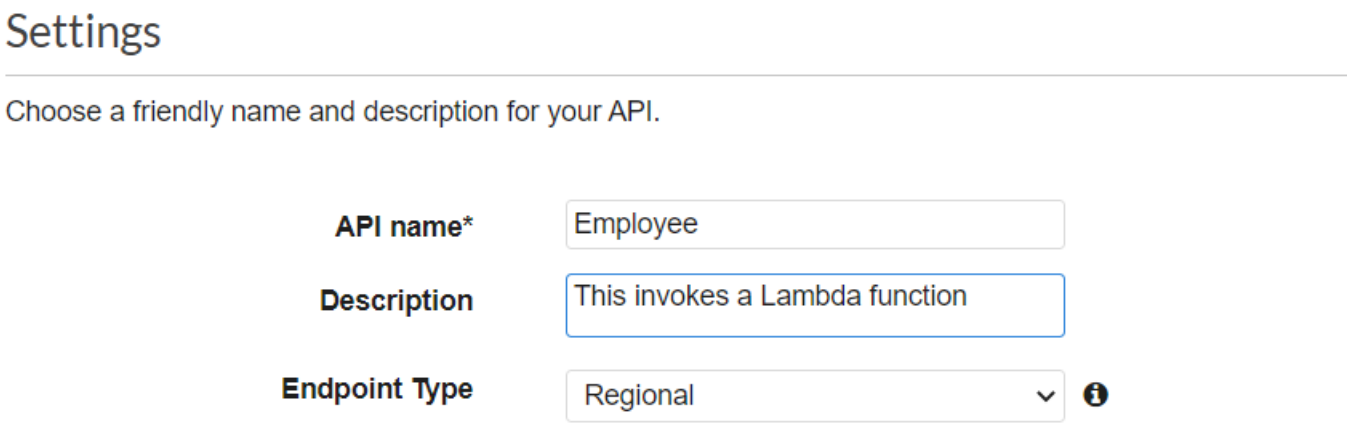
Você pode usar o console do API Gateway para criar um endpoint rest para a função do Lambda. Depois de concluído, você pode invocar a função do Lambda usando uma chamada restful.

1. Inicie uma sessão no console do Amazon API Gateway.

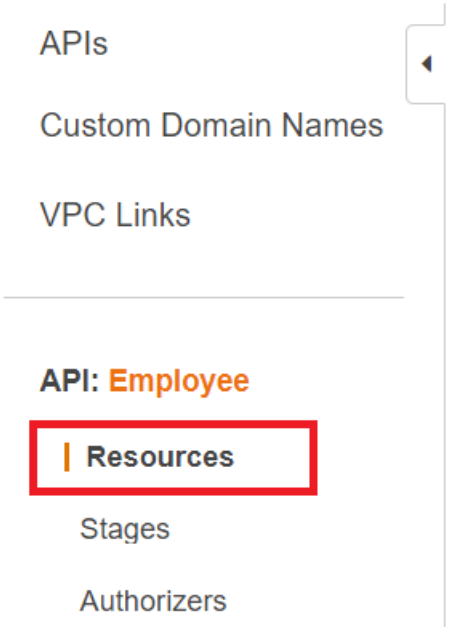
2. Na API REST, escolha Criar.
3. Selecione Nova API.



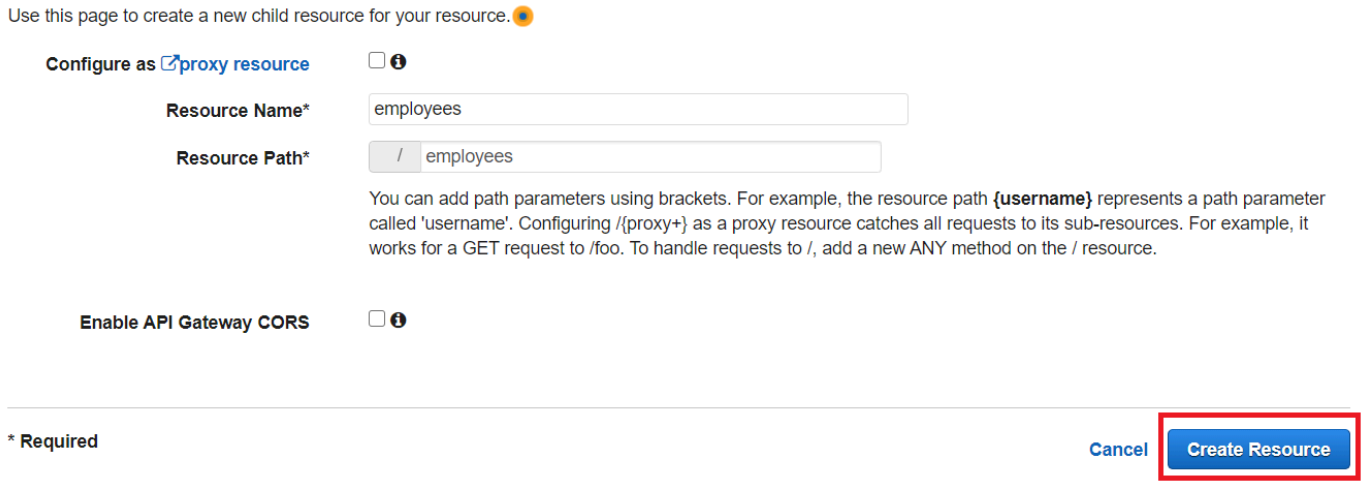
4. Especifique Funcionário como o nome da API e forneça uma descrição.



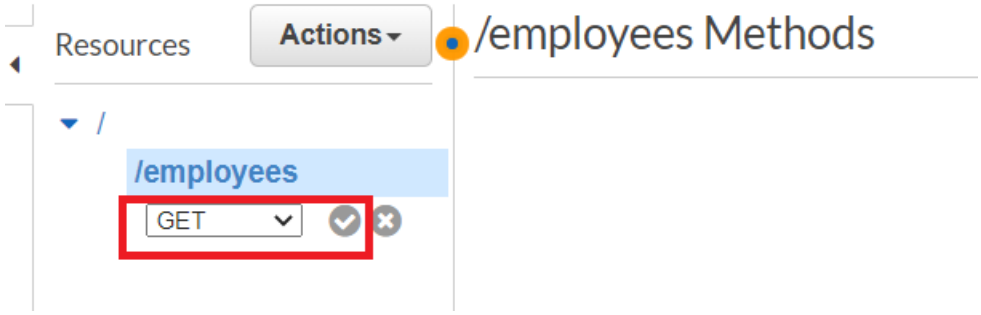
5. Selecione Criar API.
6. Escolha Recursos na seção Funcionário.



7. No campo de nome, especifique employees.
8. Selecione Create Resources (Criar recursos).
9. No menu suspenso Ações, escolha Criar recursos.



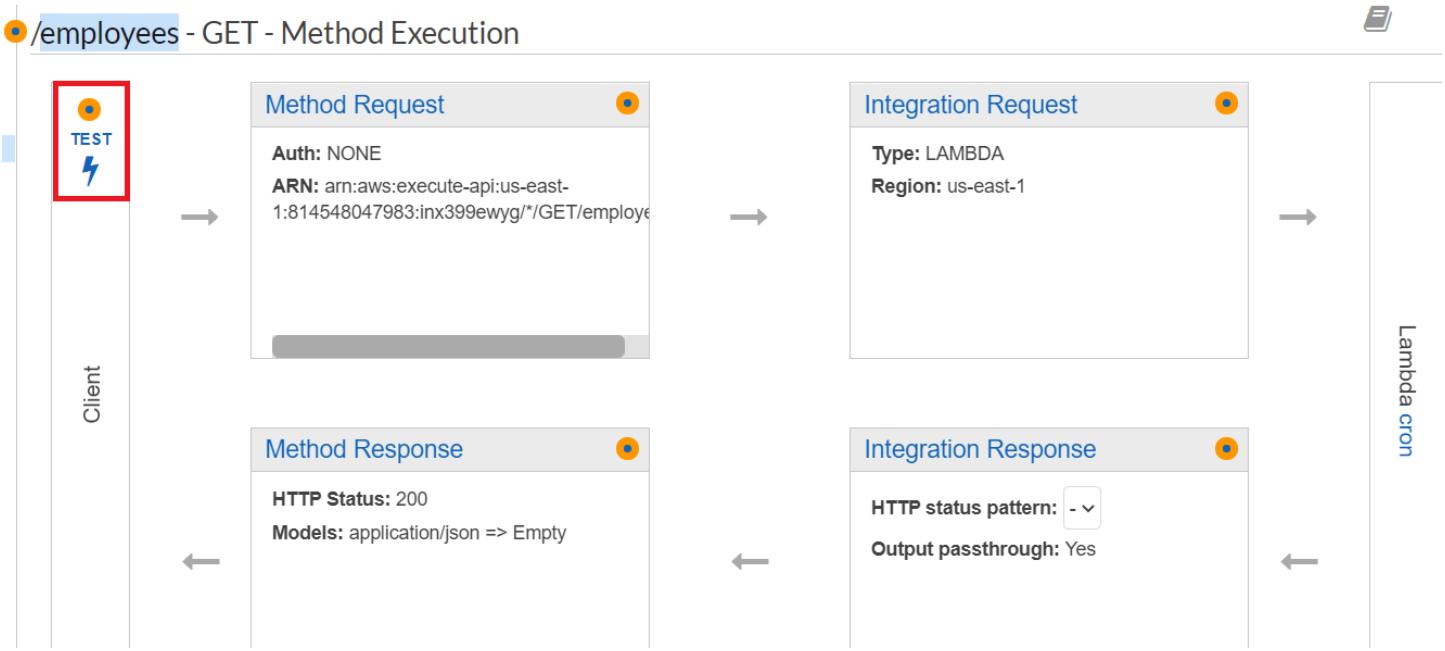
10. Escolha /employees, selecione Criar método no menu Ações e selecione GET no menu suspenso abaixo de /employees. Selecione o ícone de marca de seleção.



11. Escolha Função do Lambda e insira mylambdafunction como o nome da função do Lambda. Escolha Save (Salvar).

Testar o método do API Gateway

Neste ponto do tutorial, você pode testar o método do API Gateway que invoca a função do Lambda mylambdafunction. Para testar o método, escolha Testar, conforme mostrado na ilustração a seguir.

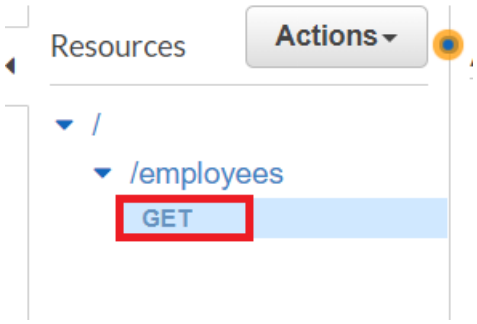


Depois que a função do Lambda é invocada, você pode visualizar o arquivo de log para ver uma mensagem de sucesso.

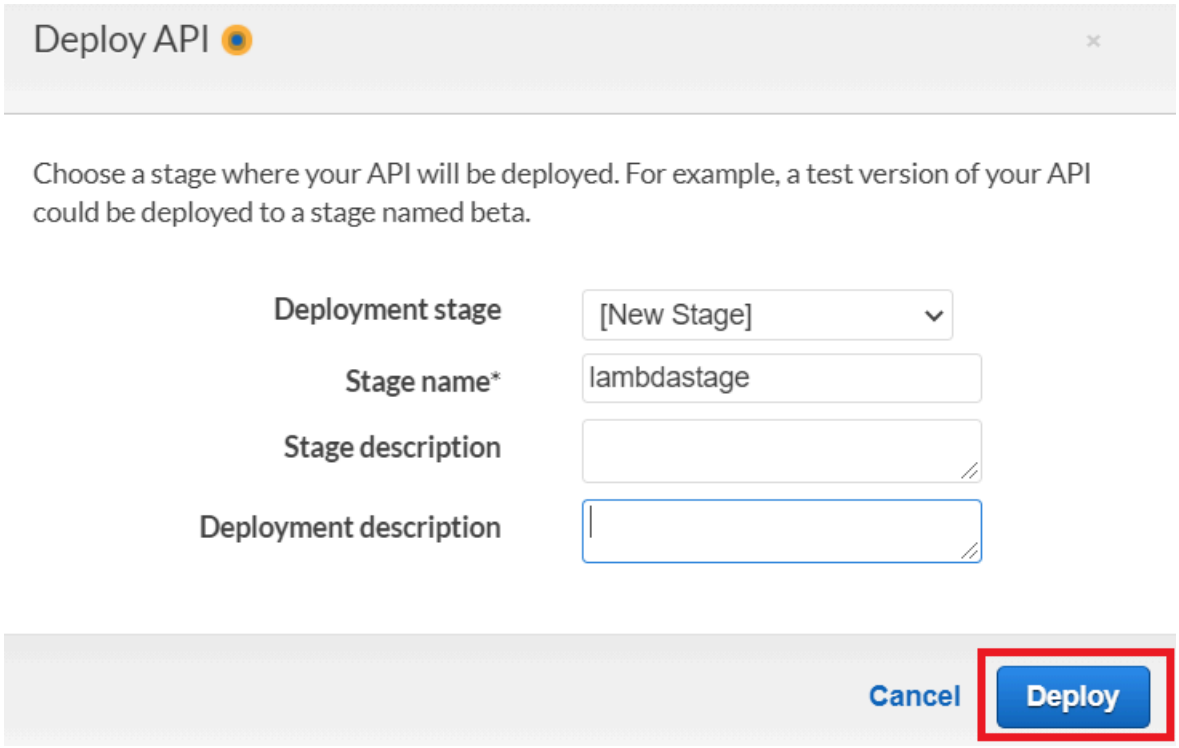
Implantar o método do API Gateway

Após o teste bem-sucedido, você pode implantar o método do console do Amazon API Gateway.

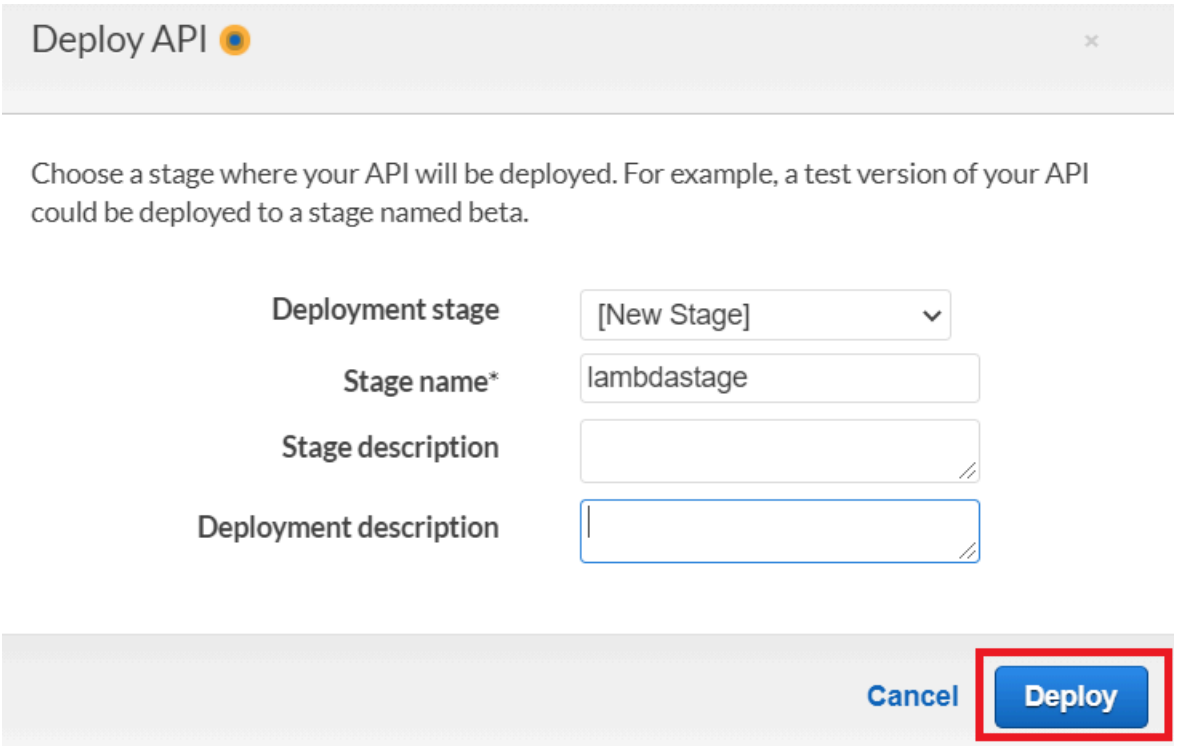
1. Selecione GET.



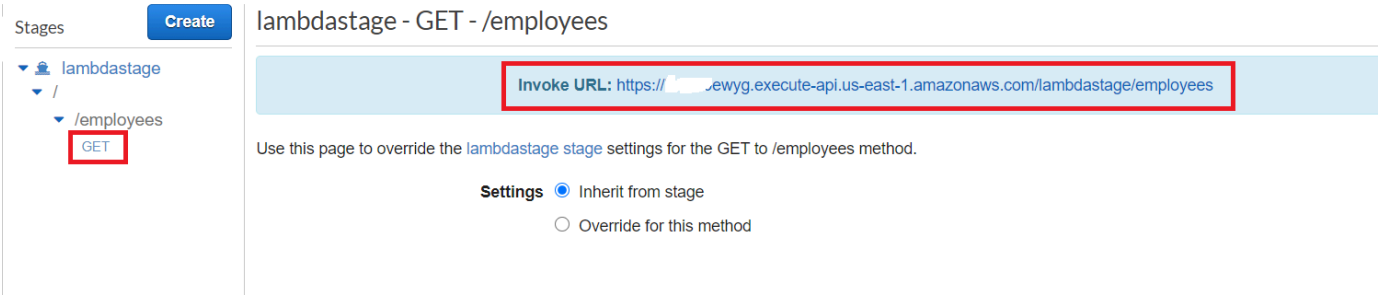
2. No menu suspenso Ações, selecione Implantar API.



3. Preencha o formulário Implantar API e escolha Implantar.



4. Escolha Save Changes (Salvar alterações).
5. Escolha Get novamente e observe que o URL muda. Esse é o URL de invocação que você pode usar para invocar a função do Lambda.



Excluir os recursos

Parabéns! Você invocou uma função do Lambda por meio do Amazon API Gateway usando o AWS SDK for JavaScript. Conforme informado no início deste tutorial, certifique-se de encerrar todos os recursos que criar enquanto percorre este tutorial para garantir que você não seja cobrado. Você pode fazer isso excluindo a pilha AWS CloudFormation que criou no tópico Criar os recursos da AWS deste tutorial, da seguinte forma:

1. Abra o AWS CloudFormation no console de gerenciamento da AWS.

2. Abra a página Pilhas e selecione a pilha.
3. Escolha Delete (Excluir).

Criação de fluxos de trabalho sem servidor da AWS usando o AWS SDK for JavaScript

Você pode criar um fluxo de trabalho sem servidor da AWS usando o Step Functions, o AWS SDK para Java e AWS Step Functions. 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.

> **Note**
>
> Você pode criar funções do Lambda em várias linguagens de programação. Neste tutorial, as funções do Lambda foram implementadas usando a API Lambda Java. Para obter mais informações sobre Lambda, consulte O que é o Lambda?

Neste tutorial, você cria um fluxo de trabalho que cria tíquetes de suporte para uma organização. Cada etapa do fluxo de trabalho executa uma operação no ticket. Este tutorial mostra como usar JavaScript para processar dados do fluxo de trabalho. Por exemplo, você aprenderá como ler dados que são passados para o fluxo de trabalho, como transmitir dados entre etapas e como invocar serviços da AWS do fluxo de trabalho.

Custo para concluir: os serviços da AWS incluídos neste documento estão incluídos no Nível gratuito da AWS.

Observação: certifique-se de encerrar todos os recursos que você criar ao passar por este tutorial para garantir que você não seja cobrado.

Tópicos

- Tarefas de pré-requisito
- Criar os recursos da AWS
- Criação do fluxo de trabalho
- Criar as funções do Lambda

- Adicionar as funções do Lambda aos fluxos de trabalho
- Executar seu fluxo de trabalho usando o console do Step Functions
- Excluir os recursos da AWS

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses exemplos do Node TypeScript e instale os módulos do AWS SDK for JavaScript e de terceiros necessários. Siga as instruções no GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

Criar os recursos da AWS

Este tutorial requer os seguintes recursos:

- Uma tabela do Amazon DynamoDB chamada Case com uma chave chamada Id.
- Um perfil do IAM chamado `lambda-support` usado para invocar funções do Lambda. Esse perfil tem políticas que permitem invocar os serviços Amazon DynamoDB e Amazon Simple Email Service a partir de uma função do Lambda.
- Um perfil do IAM chamado `workflow-support`, usado para invocar o fluxo de trabalho.
- Um bucket do Amazon S3 para hospedar as funções do Lambda.

Você pode criar esses recursos manualmente, mas recomendamos provisioná-los usando o AWS Cloud Development Kit (AWS CDK) (AWS CDK) conforme descrito neste tutorial.

Criar os recursos da AWS usando o AWS CloudFormation

O AWS CloudFormation permite que você crie e provisione implantações de infraestrutura da AWS de maneira previsível e repetida. Para mais informações sobre o AWS CloudFormation, consulte o AWS CloudFormationGuia do usuário do .

Para criar a pilha do AWS CloudFormation:

1. Instale e configure a AWS CLI seguindo as instruções no Guia do usuário da AWS CLI.

2. Crie um arquivo chamado `setup.yaml` no diretório raiz da pasta do projeto e copie o conteúdo aqui no GitHub para ele.

   > **Note**
   >
   > O modelo AWS CloudFormation foi gerado usando o AWS CDK disponível aqui no GitHub. Para obter mais informações sobre o AWS CDK, consulte o Guia do desenvolvedor do AWS Cloud Development Kit (AWS CDK).

3. Execute o comando a seguir na linha de comando, substituindo *`STACK_NAME`* por um nome exclusivo para a pilha.

   > **Important**
   >
   > O nome da pilha deve ser exclusivo em uma Região da AWS e em uma conta da AWS. Você pode especificar até 128 caracteres, e números e hifens são permitidos.

   ```
   aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
   setup.yaml --capabilities CAPABILITY_IAM
   ```

   Para obter mais informações sobre os parâmetros do comando `create-stack`, consulte o Guia de referência de comandos da AWS CLI e o Guia do usuário do AWS CloudFormation.

Crie os recursos da AWS usando o Console de gerenciamento do Amazon Web Services.

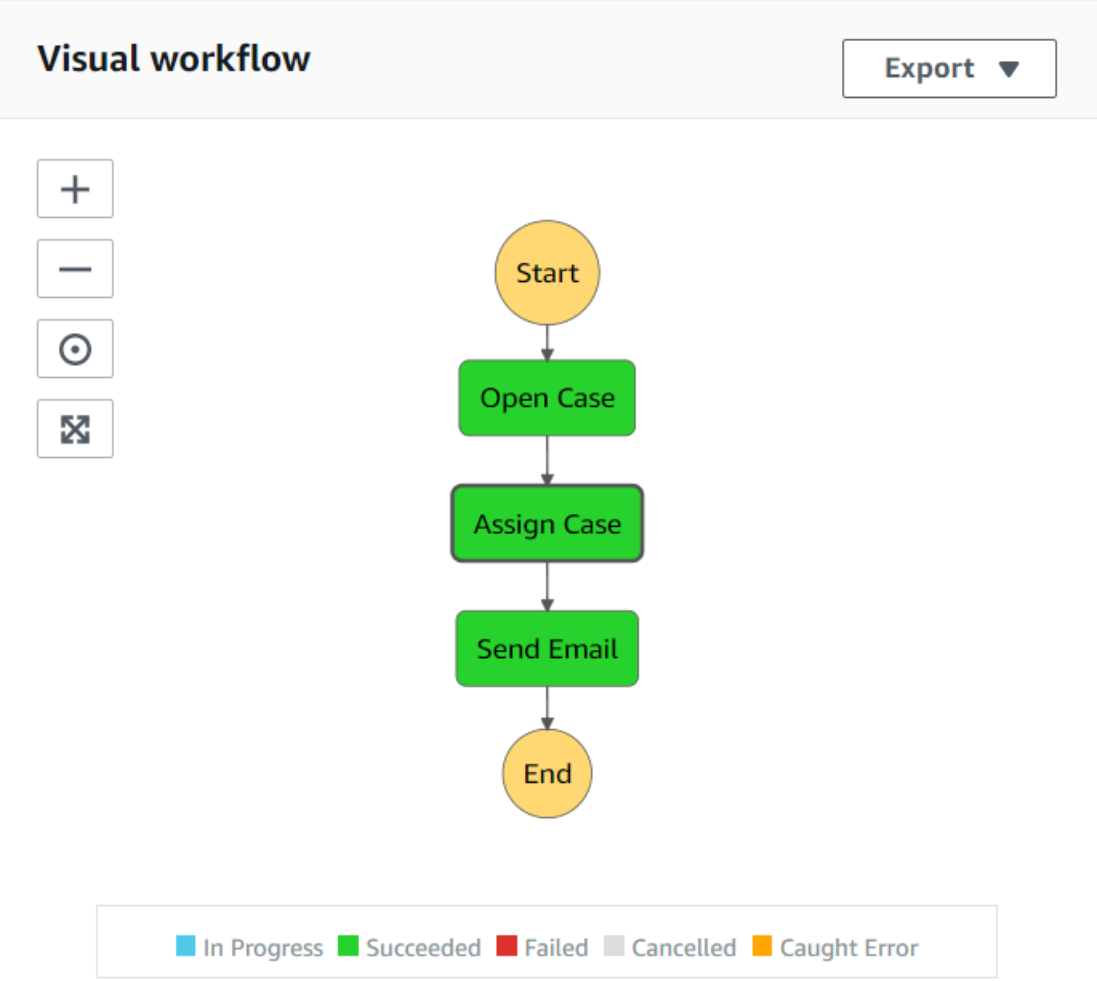
Para criar recursos para o aplicativo no console, siga as instruções no Guia do usuário do AWS CloudFormation. Use o modelo fornecido para criar um arquivo chamado `setup.yaml`, e copie o conteúdo aqui no GitHub.

> **Important**
>
> O nome da pilha deve ser exclusivo em uma Região da AWS e na conta da AWS. Você pode especificar até 128 caracteres. São permitidos números e hifens.

Veja uma lista dos recursos no console abrindo a pilha no painel AWS CloudFormation e escolhendo a guia Recursos. Você precisa desses recursos para o tutorial.

Criação do fluxo de trabalho

A figura a seguir mostra o fluxo de trabalho que você criará com este tutorial.



Veja a seguir o que acontece em cada etapa do fluxo de trabalho:

+ Início: inicia o fluxo de trabalho.

+ Abrir caso: manipula um valor de ID de tíquete de suporte passando-o para o fluxo de trabalho.

+ Atribuir caso: atribui o caso de suporte a um funcionário e armazena os dados em uma tabela do DynamoDB.

+ Enviar e-mail: envia ao funcionário uma mensagem de e-mail usando o Amazon Simple Email Service (Amazon SES) para informá-lo de que há um novo tíquete.

+ Fim: para o fluxo de trabalho.

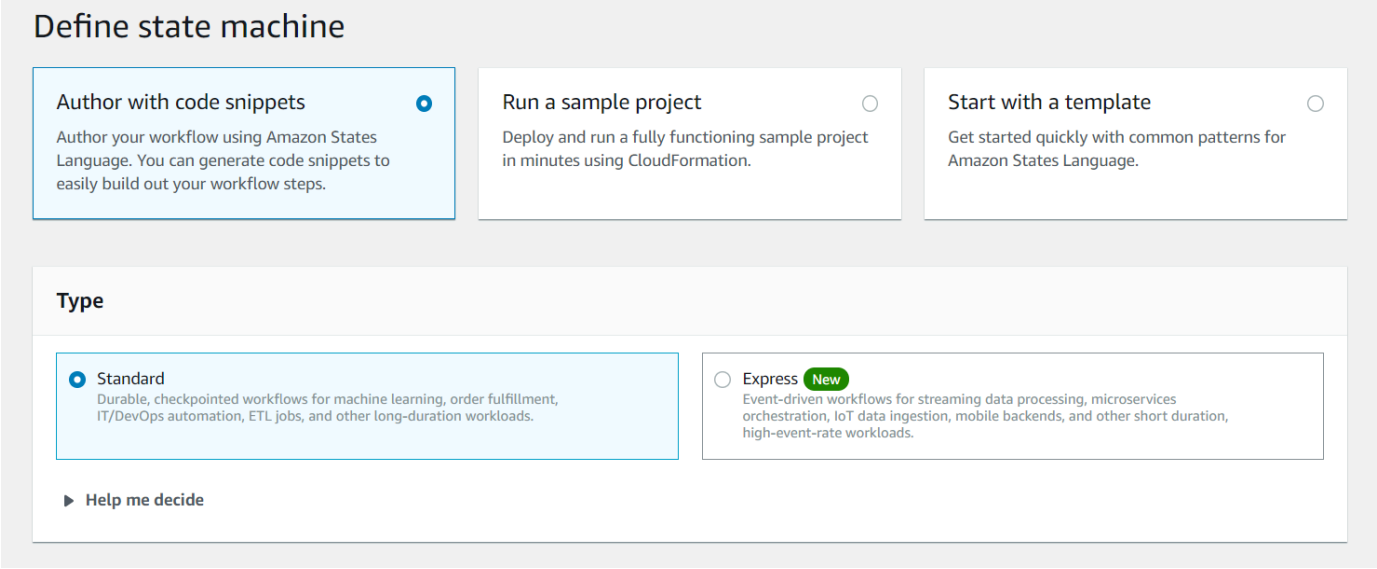
Criar um fluxo de trabalho sem servidor usando o Step Functions

É possível criar um fluxo de trabalho que processe tíquetes de suporte. Para definir um fluxo de trabalho usando o Step Functions, você cria um documento do Amazon States Language (baseado em JSON) para definir sua máquina de estado. Um documento do Amazon States Language descreve cada etapa. Depois de definir o documento, o Step Functions fornece uma representação visual do fluxo de trabalho. A figura a seguir mostra o documento do Amazon States Language e a representação visual do fluxo de trabalho.

Os fluxos de trabalho podem transmitir dados entre as etapas. Por exemplo, a etapa Abrir caso processa um valor de ID de caso (passado para o fluxo de trabalho) e passa esse valor para a etapa Atribuir caso. Mais adiante neste tutorial, você criará a lógica do aplicativo na função do Lambda para ler e processar os valores dos dados.

Para criar um fluxo de trabalho

1. Abra o Console do Amazon Web Services.
2. Escolha Create State Machine.
3. Escolha Author with code snippets (Criar com trechos de código). Na área Tipo, escolha Padrão.



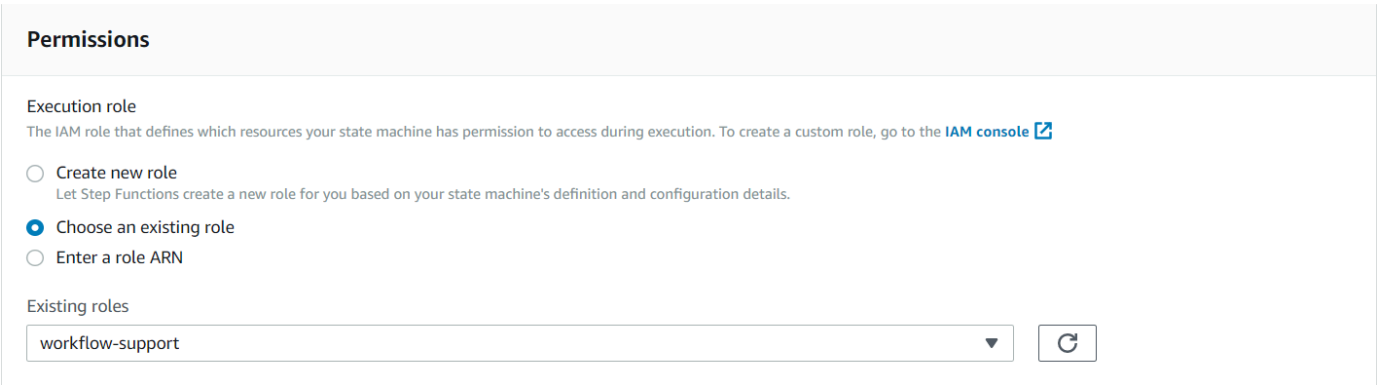
4. Especifique o documento do Amazon States Language inserindo o código a seguir.

```
{
"Comment": "A simple AWS Step Functions state machine that automates a call center
 support session.",
"StartAt": "Open Case",
"States": {
```

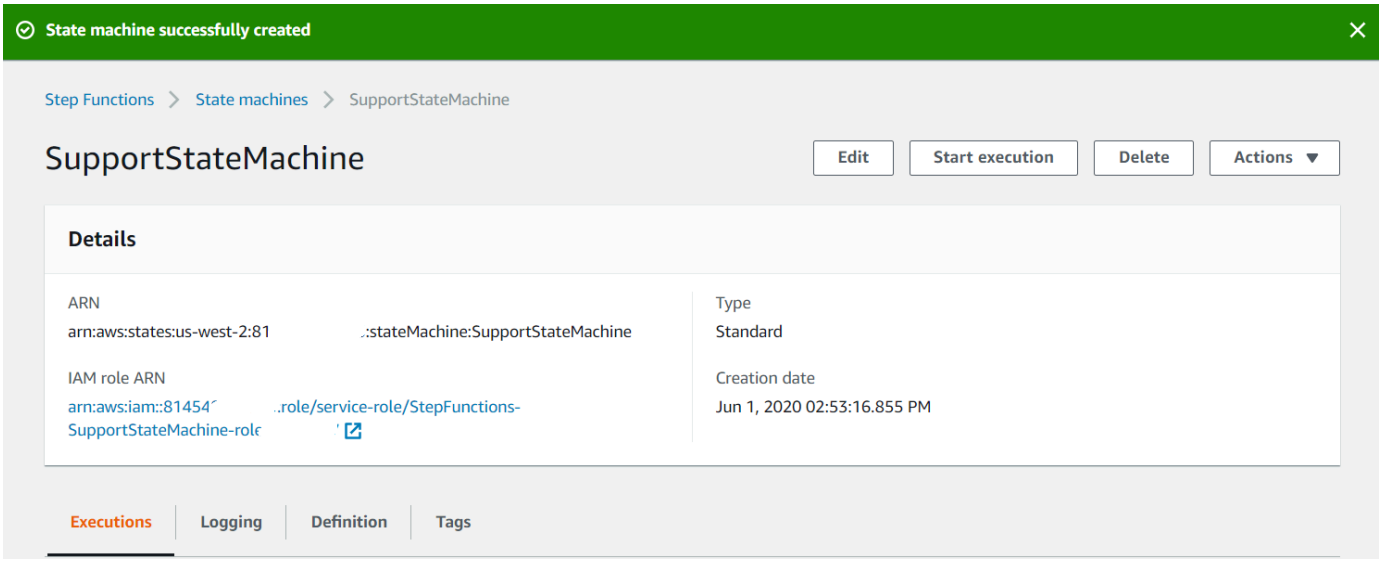
```
"Open Case": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"Next": "Assign Case"
},
"Assign Case": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"Next": "Send Email"
},
"Send Email": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"End": true
}
}
}
```

> **Note**
>
> Não se preocupe com os erros relacionados aos valores dos recursos do Lambda. Você vai atualizar esses valores mais tarde neste tutorial.

5. Escolha Next (Próximo).
6. No campo de nome, insira SupportStateMachine.
7. Na seção Permissões, selecione Escolher entre perfis do IAM existentes.
8. Escolha workflow-support (o perfil do IAM que você criou).



9. Escolha Criar uma máquina de estado. É exibida uma mensagem informando que a máquina de estado foi criada com sucesso.



Criar as funções do Lambda

Use a API de runtime do Lambda para criar as funções do Lambda. Neste exemplo, há três etapas do fluxo de trabalho, cada uma correspondendo a cada função do Lambda.

Crie estas funções do Lambda, como descrito nas seções a seguir:

- Função do Lambda getId: usada como a primeira etapa no fluxo de trabalho que processa o valor do ID do tíquete.
- Classe do Lambda addItem: usada como a segunda etapa no fluxo de trabalho que atribui o tíquete a um funcionário e armazena os dados em um banco de dados do DynamoDB.
- Classe do Lambda sendemail: usada como a terceira etapa no fluxo de trabalho que usa o Amazon SES para enviar uma mensagem de e-mail ao funcionário para notificá-lo sobre o tíquete.

Função do Lambda getId

Crie uma função do Lambda que retorne o valor do ID do tíquete que é passado para a segunda etapa no fluxo de trabalho.

```
exports.handler = async (event) => {
// Create a support case using the input as the case ID, then return a confirmation
 message
try{
    const myCaseID = event.inputCaseID;
```

```
    var myMessage = "Case " + myCaseID + ": opened...";
    var result = { Case: myCaseID, Message: myMessage };
   }
catch(err){
    console.log('Error', err);
   }
};
```

Digite o seguinte na linha de comando para usar o webpack para empacotar o arquivo com o nome `index.js`.

```
webpack getid.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

Depois, compacte `index.js` em um arquivo `ZIP` chamado `getid.js.zip`. Faça upload do arquivo `ZIP` para o bucket do Amazon S3 que você criou no tópico deste exemplo.

Este exemplo de código está disponível aqui no GitHub.

Classe do Lambda addItem

Crie uma função do Lambda que selecione um funcionário para atribuir o tíquete, e armazene os dados do tíquete em uma tabela do DynamoDB chamada Caso.

```
"use strict";
// Load the required clients and commands.
const { PutItemCommand } = require ( "@aws-sdk/client-dynamodb" );
const { dynamoClient } = require ( "../libs/dynamoClient" );

exports.handler = async (event) => {
  try {
    // Helper function to send message using Amazon SNS.
    const val = event;
    //PersistCase adds an item to a DynamoDB table
    const tmp = Math.random() <= 0.5 ? 1 : 2;
    console.log(tmp);
    if (tmp == 1) {
      const params = {
        TableName: "Case",
        Item: {
          id: { N: val.Case },
```

```
          empEmail: { S: "brmur@amazon.com" },
          name: { S: "Tom Blue" },
        },
      };
      console.log("adding item for tom");
      try {
        const data = await dynamoClient.send(new PutItemCommand(params));
        console.log(data);
      } catch (err) {
        console.error(err);
      }
      var result = { Email: params.Item.empEmail };
      return result;
    } else {
      const params = {
        TableName: "Case",
        Item: {
          id: { N: val.Case },
          empEmail: { S: "RECEIVER_EMAIL_ADDRESS" }, // Valid Amazon Simple
 Notification Services (Amazon SNS) email address.
          name: { S: "Sarah White" },
        },
      };
      console.log("adding item for sarah");
      try {
        const data = await dynamoClient.send(new PutItemCommand(params));
        console.log(data);
      } catch (err) {
        console.error(err);
      }
      return params.Item.empEmail;
      var result = { Email: params.Item.empEmail };
    }
  } catch (err) {
    console.log("Error", err);
  }
};
```

Digite o seguinte na linha de comando para usar o webpack para empacotar o arquivo com o nome `index.js`.

```
webpack additem.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

Depois, compacte `index.js` em um arquivo `ZIP` chamado `additem.js.zip`. Faça upload do arquivo `ZIP` para o bucket do Amazon S3 que você criou no tópico deste exemplo.

Este exemplo de código está disponível aqui no GitHub.

Classe do Lambda sendemail

Crie uma função do Lambda que envie um e-mail para notificá-los sobre o novo tíquete. O endereço de e-mail que é passado da segunda etapa é usado.

```
// Load the required clients and commands.
const { SendEmailCommand } = require ( "@aws-sdk/client-ses" );
const { sesClient } = require ( "../libs/sesClient" );

exports.handler = async (event) => {
  // Enter a sender email address. This address must be verified.
  const senderEmail = "SENDER_EMAIL"
  const sender = "Sender Name <" + senderEmail + ">";

  // AWS Step Functions passes the  employee's email to the event.
  // This address must be verified.
  const recepient = event.S;

  // The subject line for the email.
  const subject = "New case";

  // The email body for recipients with non-HTML email clients.
  const body_text =
    "Hello,\r\n" + "Please check the database for new ticket assigned to you.";

  // The HTML body of the email.
  const body_html = `<html><head></head><body><h1>Hello!</h1><p>Please check the
 database for new ticket assigned to you.</p></body></html>`;

  // The character encoding for the email.
  const charset = "UTF-8";
  var params = {
    Source: sender,
    Destination: {
      ToAddresses: [recepient],
    },
    Message: {
      Subject: {
```

```
        Data: subject,
        Charset: charset,
      },
      Body: {
        Text: {
          Data: body_text,
          Charset: charset,
        },
        Html: {
          Data: body_html,
          Charset: charset,
        },
      },
    },
  };
  try {
    const data = await sesClient.send(new SendEmailCommand(params));
    console.log(data);
  } catch (err) {
    console.error(err);
  }
};
```

Digite o seguinte na linha de comando para usar o webpack para empacotar o arquivo com o nome `index.js`.

```
webpack sendemail.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

Depois, compacte `index.js` em um arquivo `ZIP` chamado `sendemail.js.zip`. Faça upload do arquivo `ZIP` para o bucket do Amazon S3 que você criou no tópico deste exemplo.

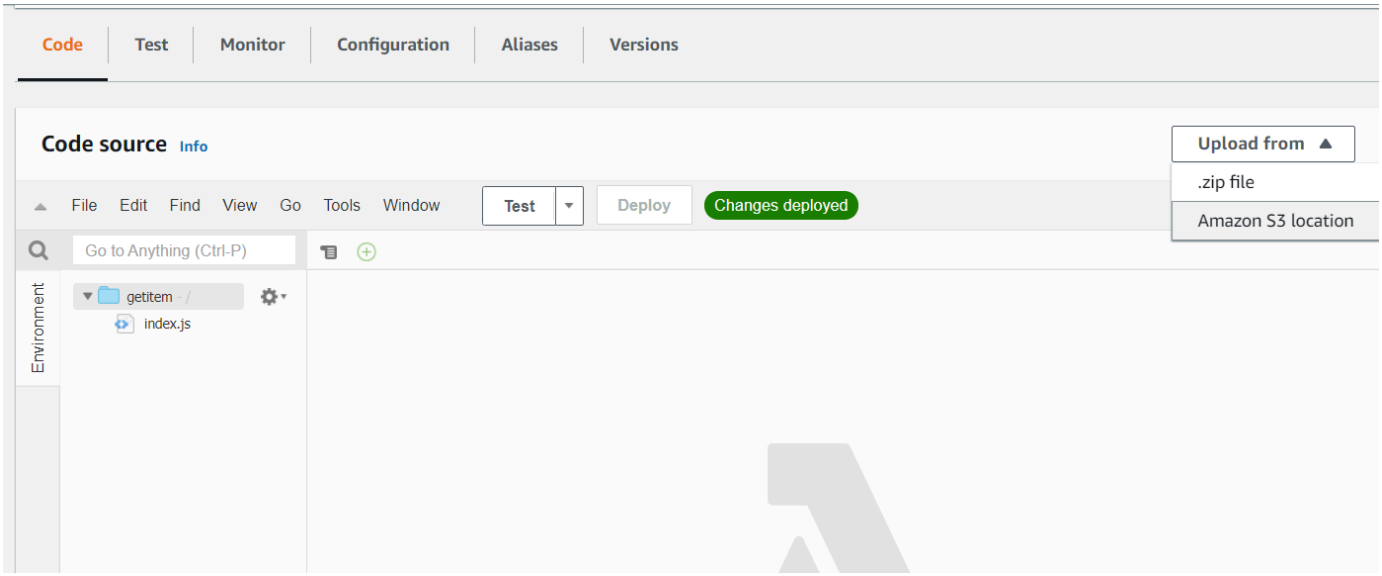
Este exemplo de código está disponível aqui no GitHub.

Implantar as funções do Lambda

Para implantar a função do Lambda `getid`:

1. Abra o console do Lambda no Console do Amazon Web Services.
2. Escolha Criar função.
3. Escolha Author from scratch (Criar do zero).

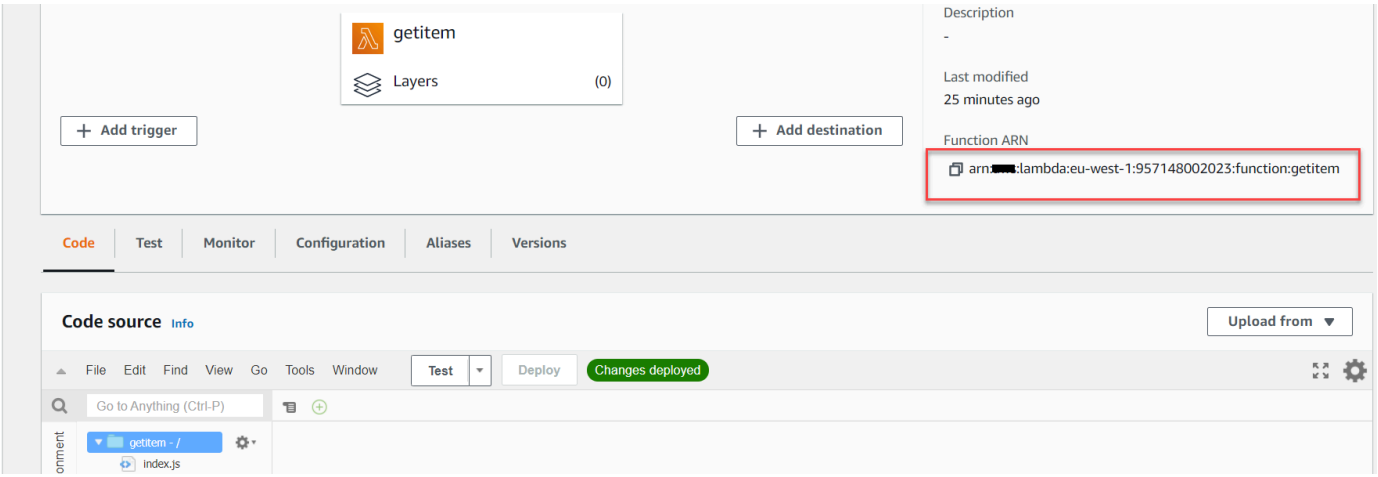
4. Na seção de informações Básicas, insira getid como o nome.
5. Em Runtime, escolha Node.js 14.x.
6. Escolha Usar uma função existente e, em seguida, escolha lambda-support (o perfil do IAM que você criou).
7. Escolha a opção Criar função.
8. Escolha Fazer upload de - localização do Amazon S3.
9. Escolha Upload, escolha Fazer upload de - localização do Amazon S3 e insira o URL do link do Amazon S3.



10. Escolha Save (Salvar).
11. Repita esse procedimento para os arquivos additem.js.zip e sendemail.js.zip para novas funções do Lambda. Quando terminar, você terá três funções do Lambda que podem ser consultadas no documento Amazon States Language.

Adicionar as funções do Lambda aos fluxos de trabalho

1. Abra o console do lambda. Observe que você pode visualizar o valor do nome do recurso da Amazon (ARN) do Lambda no canto superior direito.



2. Copie o valor e cole-o na etapa 1 do documento Amazon States Language, localizado no console do Step Functions.
3. Atualize o recurso para as etapas Atribuir caso e Enviar e-mail. É assim que você conecta as funções do Lambda criadas usando o AWS SDK para Java a um fluxo de trabalho criado usando o Step Functions.

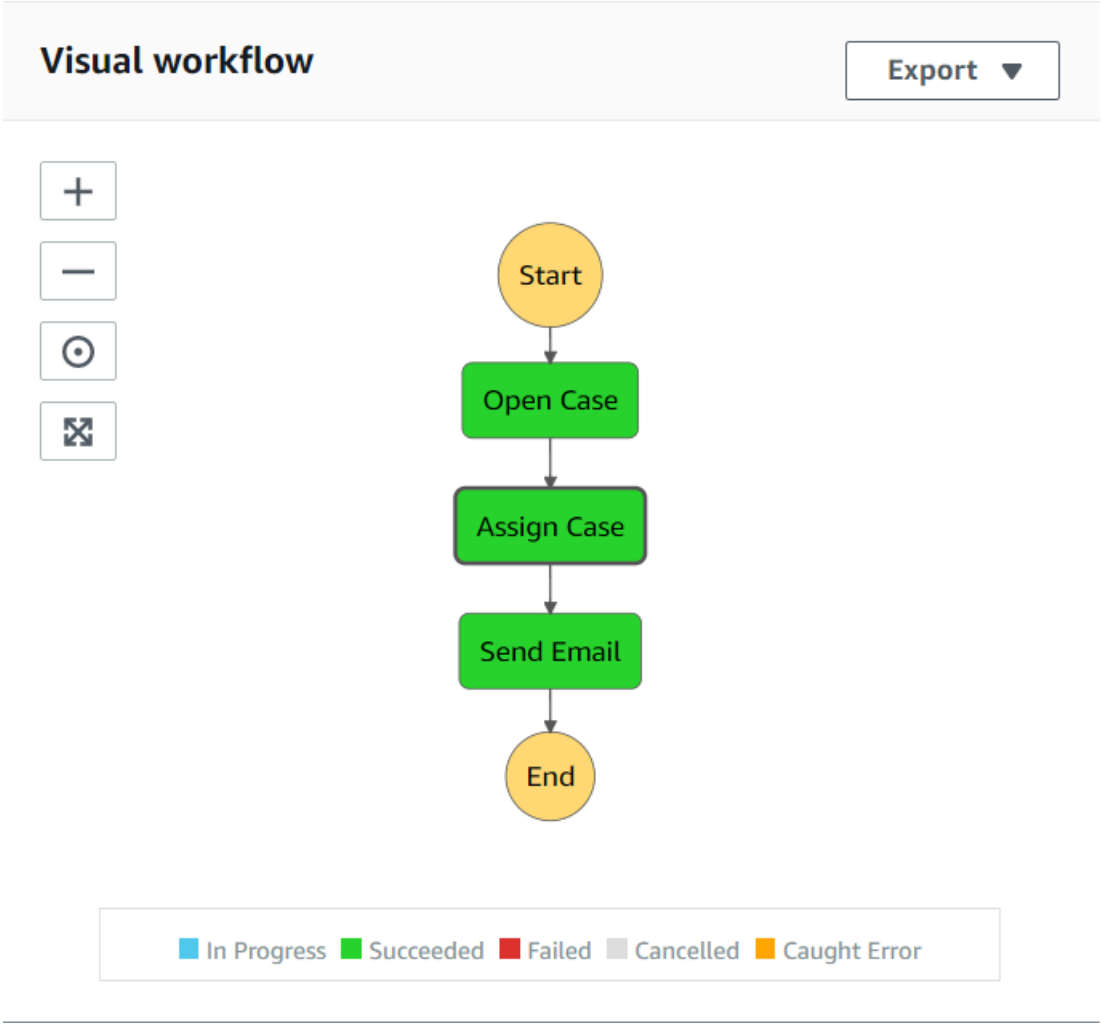
Executar seu fluxo de trabalho usando o console do Step Functions

Você pode invocar o fluxo de trabalho no console do Step Functions. Uma execução recebe a entrada JSON. Neste exemplo, você pode passar os seguintes dados JSON para o fluxo de trabalho.

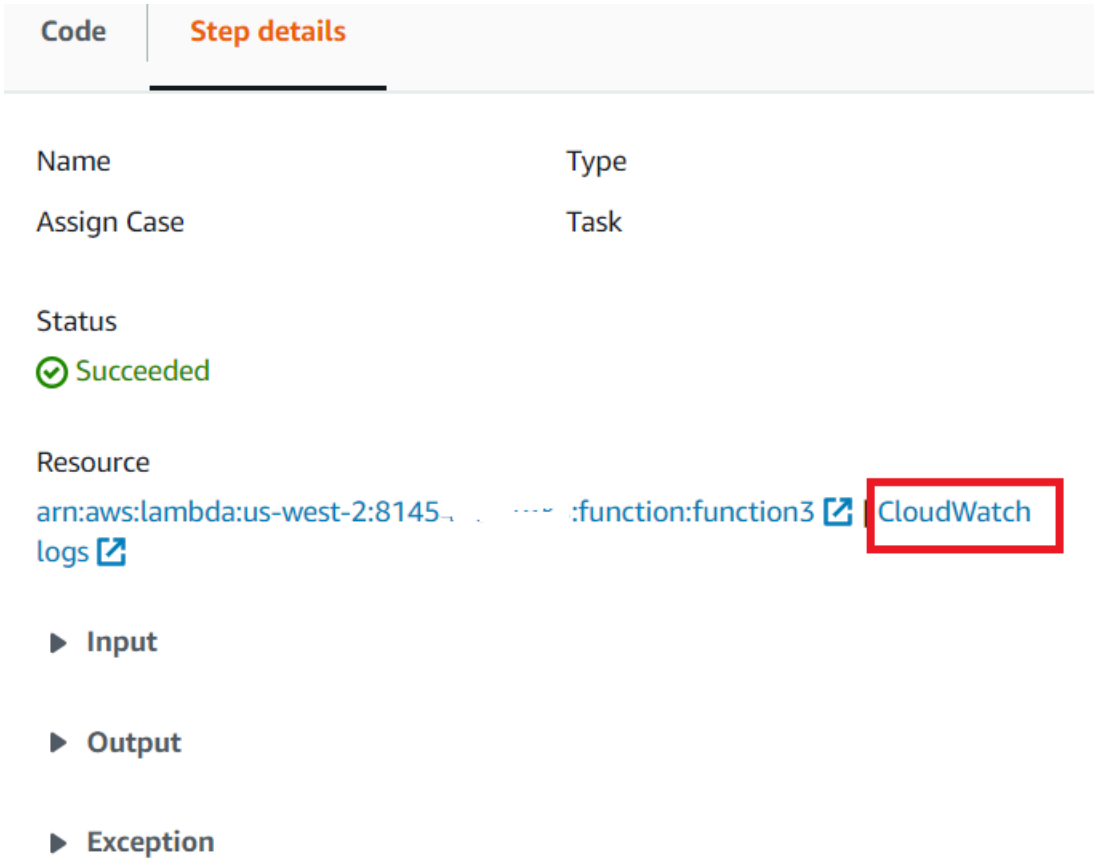
```
{
"inputCaseID": "001"
}
```

Para executar o fluxo de trabalho:

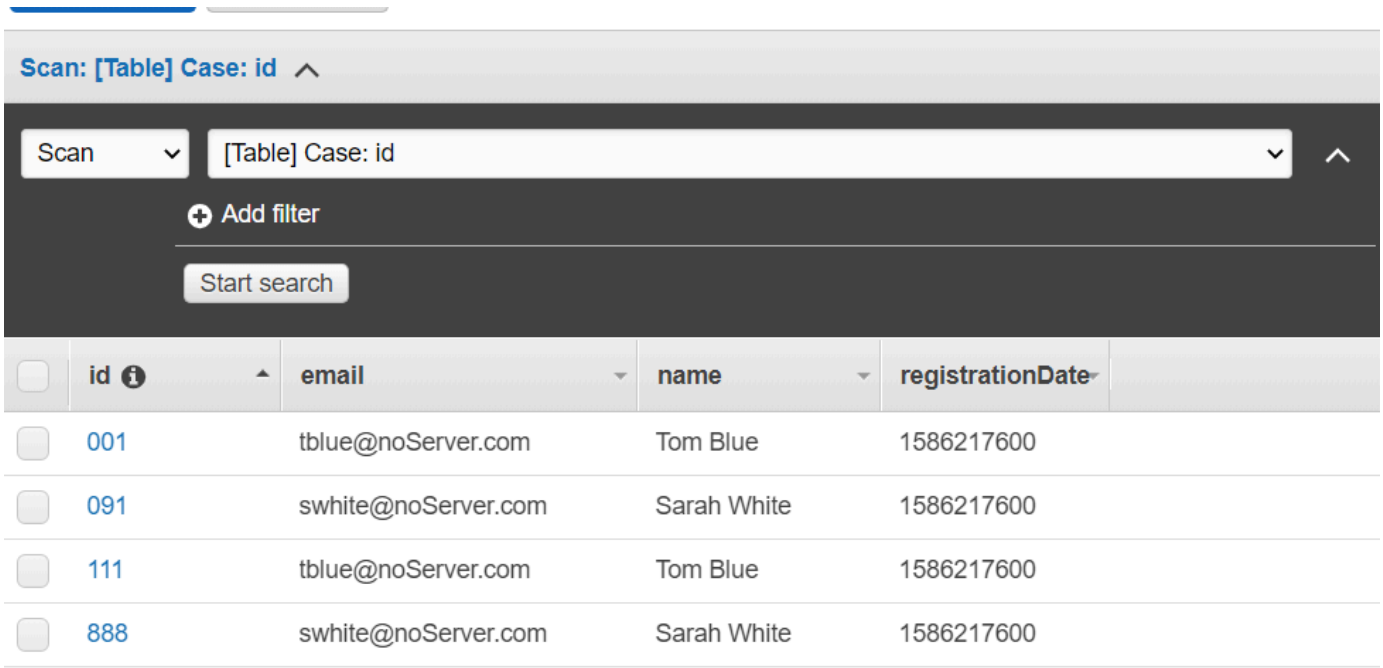
1. No console do Step Functions, escolha Iniciar execução.
2. Na seção Entrada, passe os dados JSON. Visualize o fluxo de trabalho. À medida que cada etapa é concluída, ela fica verde.



3. Se a etapa ficar vermelha, ocorreu um erro. Você pode clicar na etapa e ver os logs que podem ser acessados do lado direito.



Quando o fluxo de trabalho estiver concluído, você poderá visualizar os dados na tabela do DynamoDB.



Excluir os recursos da AWS

Parabéns, você criou um fluxo de trabalho sem servidor da AWS usando o AWS SDK para Java. Conforme informado no início deste tutorial, certifique-se de encerrar todos os recursos que criar enquanto percorre este tutorial para garantir que você não seja cobrado. Você pode fazer isso excluindo a pilha AWS CloudFormation que criou no tópico Criar os recursos da AWS deste tutorial, da seguinte forma:

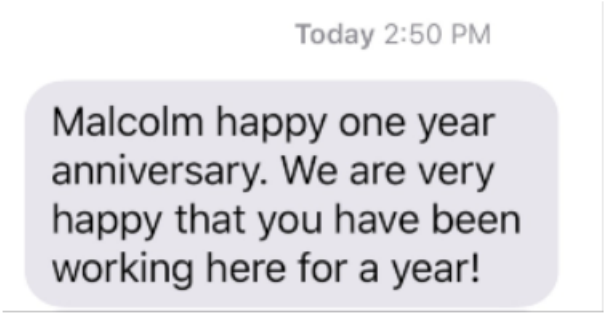
1. Abra o AWS CloudFormation no console de gerenciamento da AWS.
2. Abra a página Pilhas e selecione a pilha.
3. Escolha Delete (Excluir).

Criação de eventos programados para executar funções do AWS Lambda

Você pode criar um evento agendado que invoca uma AWS Lambda função usando um evento da Amazon CloudWatch . Você pode configurar um CloudWatch evento para usar uma expressão cron para agendar quando uma função Lambda é invocada. Por exemplo, você pode agendar um CloudWatch evento para invocar uma função Lambda todos os dias da semana.

O AWS Lambda é um serviço de computação que permite executar código sem provisionar ou gerenciar servidores. Você pode criar funções do Lambda em várias linguagens de programação. Para obter mais informações sobre o AWS Lambda, consulte O que é o AWS Lambda.

Neste tutorial, você cria uma função Lambda usando a API de tempo de execução do JavaScript Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Por exemplo, suponha que uma organização envie uma mensagem de SMS para seus funcionários parabenizando-os pela data de um ano de tempo de casa, conforme mostrado nesta ilustração.

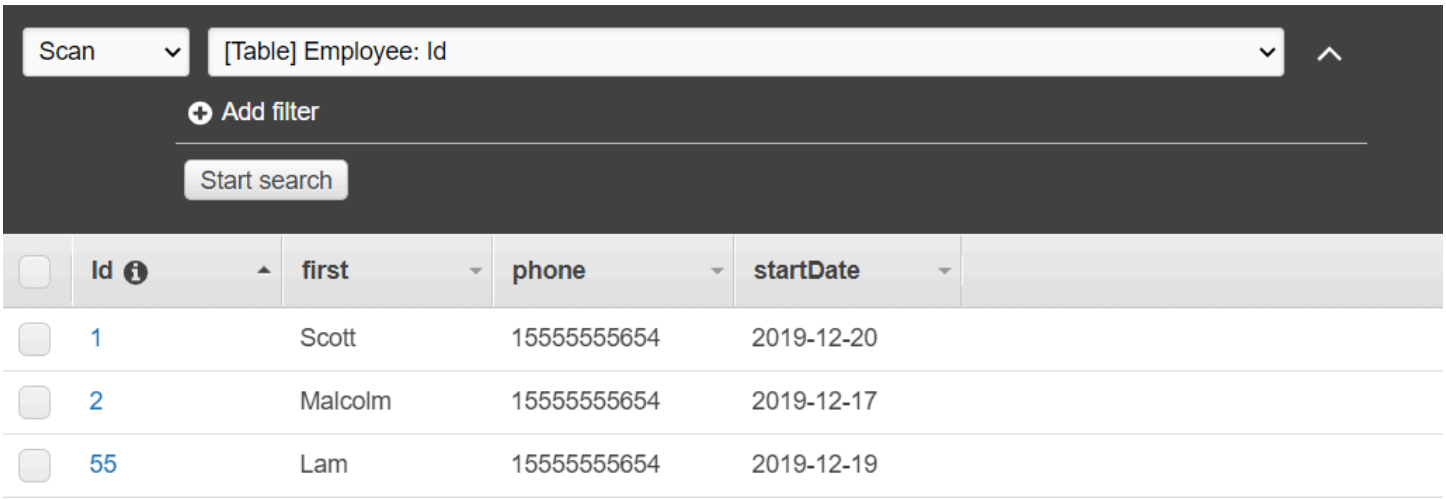


O tutorial levará aproximadamente 20 minutos para ser concluído.

Este tutorial mostra como usar a JavaScript lógica para criar uma solução que execute esse caso de uso. Por exemplo, você aprenderá como ler um banco de dados para determinar quais funcionários atingiram a data de um ano de tempo de casa, como processar os dados e enviar uma mensagem de texto usando uma função do Lambda. Em seguida, você aprenderá como usar uma expressão cron para invocar a função do Lambda todos os dias da semana.

Este tutorial da AWS usa uma tabela do Amazon DynamoDB chamada Funcionários que contém estes campos.

- id - a chave primária da tabela.
- firstName - o nome do funcionário.
- phone - o número de telefone do funcionário.
- startDate - a data de início do funcionário.



> **Important**
>
> Custo para concluir: os serviços da AWS incluídos neste documento estão incluídos no Nível gratuito da AWS. No entanto, certifique-se de encerrar todos os recursos depois de concluir este tutorial para garantir que você não seja cobrado.

Para criar o aplicativo:

1. Concluir os pré-requisitos
2. Criar os recursos da AWS

3. Prepare o script do navegador
4. Criar e carregar a função do Lambda
5. Implantar a função do Lambda
6. Executar o aplicativo
7. Excluir os recursos

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses TypeScript exemplos de Node.js e instale os módulos necessários AWS SDK for JavaScript e de terceiros. Siga as instruções em GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

Criar os recursos da AWS

Este tutorial requer os seguintes recursos:

- Uma tabela do Amazon DynamoDB chamada Funcionários com uma chave chamada Id e os campos mostrados na ilustração anterior. Certifique-se de inserir os dados corretos, incluindo um telefone celular válido com o qual você deseja testar esse caso de uso. Para obter mais informações, consulte Criar uma tabela.
- Um perfil do IAM com permissões anexadas para executar funções do Lambda.
- Um bucket do Amazon S3 para hospedar a função do Lambda.

Você pode criar esses recursos manualmente, mas recomendamos provisioná-los usando o AWS CloudFormation conforme descrito neste tutorial.

Criar os recursos da AWS usando o AWS CloudFormation

O AWS CloudFormation permite que você crie e provisione implantações de infraestrutura da AWS de maneira previsível e repetida. Para mais informações sobre o AWS CloudFormation, consulte o AWS CloudFormationGuia do usuário do .

Para criar a pilha AWS CloudFormation usando a AWS CLI:

1. Instale e configure a AWS CLI seguindo as instruções do Guia do usuário da AWS CLI.
2. Crie um arquivo chamado `setup.yaml` no diretório raiz da pasta do seu projeto e copie o conteúdo aqui GitHub para dentro dele.

   > **Note**
   >
   > O AWS CloudFormation modelo foi gerado usando o AWS CDK disponível aqui em GitHub. Para obter mais informações sobre o AWS CDK, consulte o Guia do desenvolvedor do AWS Cloud Development Kit (AWS CDK).

3. Execute o comando a seguir na linha de comando, substituindo *`STACK_NAME`* por um nome exclusivo para a pilha.

   > **Important**
   >
   > O nome da pilha deve ser exclusivo em uma Região da AWS e em uma conta da AWS. Você pode especificar até 128 caracteres, e números e hifens são permitidos.

   ```
   aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
   setup.yaml --capabilities CAPABILITY_IAM
   ```

   Para obter mais informações sobre os parâmetros do comando `create-stack`, consulte o Guia de referência de comandos da AWS CLI e o Guia do usuário do AWS CloudFormation.

   Veja uma lista dos recursos no console abrindo a pilha no painel AWS CloudFormation e escolhendo a guia Recursos. Você precisa desses recursos para o tutorial.
4. Quando a pilha for criada, use o AWS SDK for JavaScript para preencher a tabela do DynamoDB, conforme descrito em Preencher a tabela do DynamoDB..

Preencher a tabela do DynamoDB.

Para preencher a tabela, primeiro crie um diretório chamado `libs`, nele crie um arquivo chamado `dynamoClient.js` e cole o conteúdo abaixo nesse arquivo.

```
const { DynamoDBClient } = require( "@aws-sdk/client-dynamodb" );
```

```
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon DynamoDB service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

Esse código está disponível aqui em GitHub.

Em seguida, crie um arquivo chamado `populate-table.js` no diretório raiz da pasta do seu projeto e copie o conteúdo aqui GitHub para dentro dele. Para um dos itens, substitua o valor da propriedade phone por um número de celular válido no formato E.164, e o valor de `startDate` pela data de hoje.

Na linha de comando, execute o seguinte comando:

```
node populate-table.js
```

```
const {
BatchWriteItemCommand } = require( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require(  "./libs/dynamoClient" );
// Set the parameters.
const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
```

```
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Esse código está disponível aqui em GitHub.

Criar a função do AWS Lambda

Como configurar o SDK

Primeiro, importe os módulos e comandos necessários do AWS SDK for JavaScript (v3): `DynamoDBClient` e DynamoDB `ScanCommand`, e `SNSClient` e Amazon SNS `PublishCommand`. Substitua *`REGION`* pela Região da AWS. Em seguida, calcule a data de hoje e atribua-a a um parâmetro. Em seguida, crie os parâmetros para `ScanCommand`.Replace *`TABLE_NAME`* com o nome da tabela que você criou na seção Criar os recursos da AWS deste exemplo.

O snippet de código a seguir mostra essa etapa. (Consulte Como empacotar a função do Lambda para ver o exemplo completo.)

```
"use strict";
```

```
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};
```

Como escanear a tabela do DynamoDB

Primeiro, crie uma função async/await chamada `sendText` para publicar uma mensagem de texto usando o Amazon SNS `PublishCommand`. Em seguida, adicione um padrão de bloco `try` que verifique a tabela do DynamoDB em busca de funcionários com aniversário de tempo de casa na data de hoje e, em seguida, chame a função `sendText` para enviar uma mensagem de texto a esses funcionários. Se ocorrer um erro, o bloco `catch` será chamado.

O snippet de código a seguir mostra essa etapa. (Consulte Como empacotar a função do Lambda para ver o exemplo completo.)

```
exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
```

```
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Como empacotar a função do Lambda

Este tópico descreve como empacotar a `mylambdafunction.js` e os módulos do AWS SDK for JavaScript necessários para este exemplo em um arquivo empacotado chamado `index.js`.

1. Caso ainda não tenha feito isso, siga as instruções descritas em Tarefas de pré-requisito para este exemplo para instalar o webpack.

   > **Note**
   >
   > Para obter informações sobre webpack, consulte Agrupe aplicativos com o webpack.

2. Execute o seguinte na linha de comando para agrupar o deste JavaScript exemplo em um arquivo chamado`<index.js>`:

   ```
   webpack mylamdbafunction.js --mode development --target node --devtool false --
   output-library-target umd -o index.js
   ```

> **Important**
>
> A saída é chamada `index.js`. Isso ocorre porque as funções do Lambda precisam ter um manipulador `index.js` para funcionar.

3. Comprima o arquivo de saída empacotado, `index.js`, em um arquivo ZIP chamado `my-lambda-function.zip`.
4. Faça o upload de `mylambdafunction.zip` para o bucket Amazon S3 que você criou no tópico Criar os recursos da AWS deste tutorial.

Aqui está o código completo do script do navegador para `mylambdafunction.js`.

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};
```

```
// Create the client service objects.
const dbclient = new DynamoDBClient({ region: REGION });
const snsclient = new SNSClient({ region: REGION });

exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Implante a função do Lambda.

Na raiz do projeto, crie um arquivo `lambda-function-setup.js` e cole o conteúdo abaixo nele.

Substitua *BUCKET_NAME* pelo nome do bucket Amazon S3 para o qual você fez o upload da versão ZIP da sua função do Lambda. Substitua *ZIP_FILE_NAME* pelo nome da versão ZIP da sua função do Lambda. Substitua *IAM_ROLE_ARN* pelo Número de recurso da Amazon (ARN) do perfil do IAM que você criou no tópico Criar os recursos da AWS deste tutorial. Substitua *LAMBDA_FUNCTION_NAME* por um nome para a função do Lambda.

```
// Load the required Lambda client and commands.
const {
   CreateFunctionCommand,
} = require("@aws-sdk/client-lambda");
const {
   lambdaClient
} = require("..libs/lambdaClient.js");

// Instantiate an Lambda client service object.
const lambda = new LambdaClient({ region: REGION });

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
 Services (Amazon SNS) to " +
    "send employees an email the each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambda.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

Insira o seguinte na linha de comando para implantar a função do Lambda.

```
node lambda-function-setup.js
```

Este exemplo de código está disponível aqui em GitHub.

Configure CloudWatch para invocar as funções Lambda

Para configurar para CloudWatch invocar as funções do Lambda:

1. Abra a página Functions (Funções) no console do Lambda.
2. Escolha a função Lambda.
3. Em Designer, escolha Add trigger (Adicionar trigger).
4. Defina o tipo de gatilho como CloudWatch Events/ EventBridge.
5. Em Regra, escolha Criar uma nova regra.
6. Preencha o Nome da regra e a Descrição da regra.
7. Para o tipo de regra, selecione Expressão de programação.
8. No campo Expressão de programação, insira uma expressão cron. Por exemplo, cron(0 12 ? * MON-FRI *).
9. Escolha Add.

> **Note**
>
> Para obter mais informações, consulte Usando o Lambda com CloudWatch eventos.

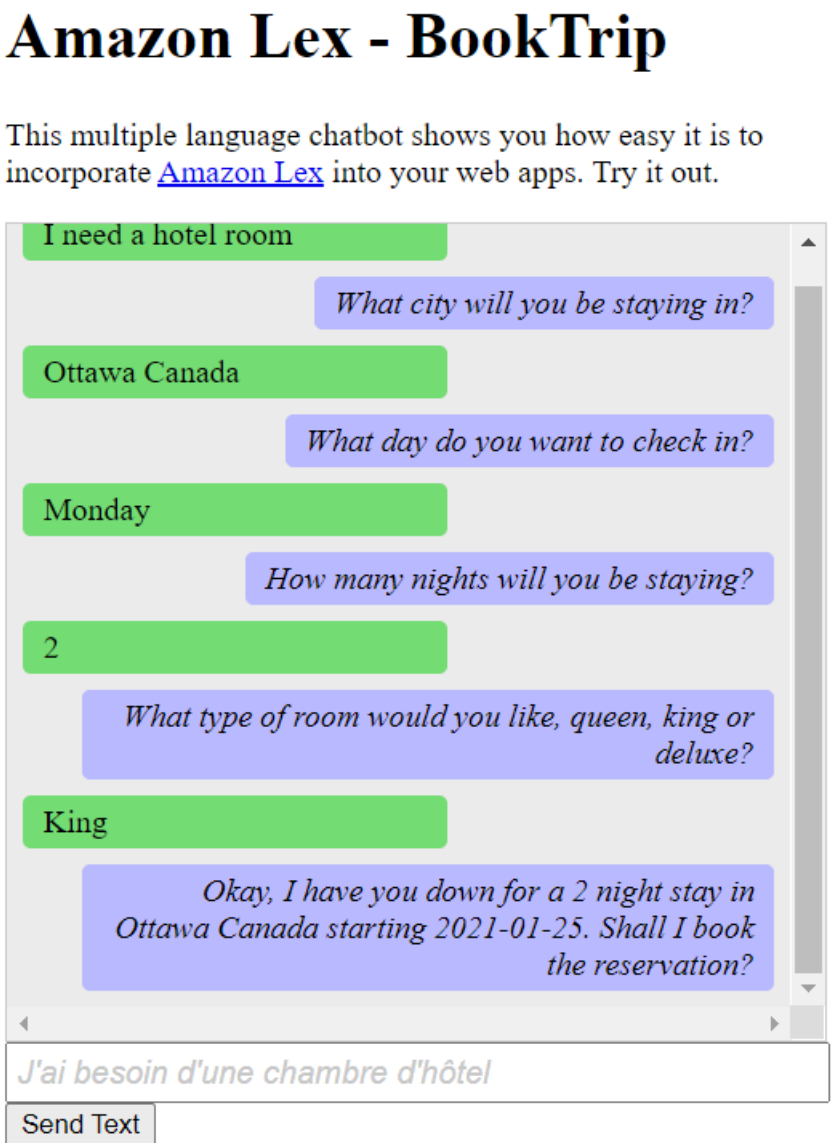
Excluir os recursos

Parabéns! Você invocou uma função Lambda por meio de eventos programados da CloudWatch Amazon usando o. AWS SDK for JavaScript Conforme informado no início deste tutorial, certifique-se de encerrar todos os recursos que criar enquanto percorre este tutorial para garantir que você não seja cobrado. Você pode fazer isso excluindo a pilha AWS CloudFormation que criou no tópico Criar os recursos da AWS deste tutorial, da seguinte forma:

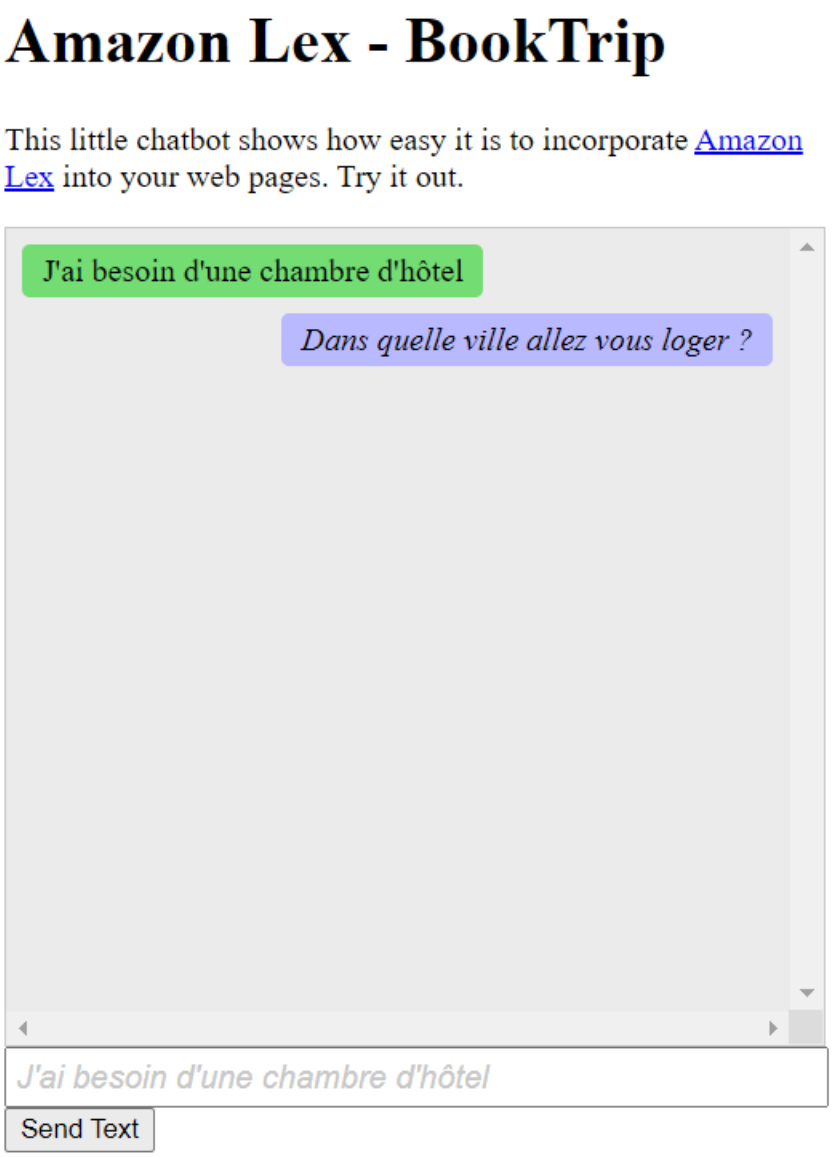
1. Abra o console de AWS CloudFormation.
2. Na página Pilhas, selecione a pilha.
3. Escolha Excluir.

Como criar um chatbot do Amazon Lex

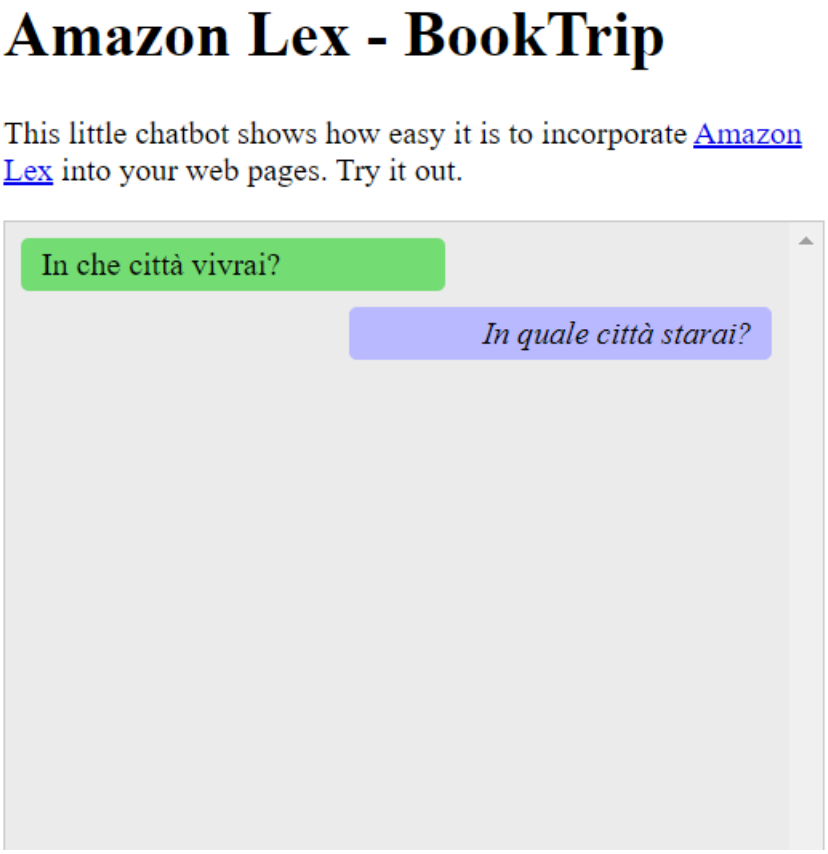
Você pode criar um chatbot do Amazon Lex em um aplicativo Web para engajar os visitantes do seu site. Um chatbot do Amazon Lex é uma funcionalidade que realiza conversas de chat online com os usuários sem fornecer contato direto com uma pessoa. Por exemplo, a ilustração a seguir mostra um chatbot do Amazon Lex que envolve um usuário para reservar um quarto de hotel.



O chatbot do Amazon Lex criado neste tutorial da AWS está acessível em vários idiomas. Por exemplo, um usuário que fala francês pode inserir um texto em francês e receber uma resposta em francês.



Da mesma forma, um usuário pode se comunicar com o chatbot do Amazon Lex em italiano.



Este tutorial da AWS orienta você na criação de um chatbot do Amazon Lex e na integração dele a um aplicativo Web Node.js. O AWS SDK for JavaScript (v3) é usado para invocar esses AWS serviços:

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

Custo para concluir: os serviços da AWS incluídos neste documento estão incluídos no Nível gratuito da AWS.

Observação: certifique-se de encerrar todos os recursos que você cria ao passar por este tutorial para garantir que você não seja cobrado.

Para criar o aplicativo:

1. Pré-requisitos
2. Provisionar recursos

3. Criar um chatbot do Amazon Lex
4. Criar o HTML
5. Criar o script do navegador
6. Próximas etapas

Pré-requisitos

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses TypeScript exemplos de Node e instale os módulos necessários AWS SDK for JavaScript e de terceiros. Siga as instruções em GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Este exemplo usa ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Criar os recursos da AWS

Este tutorial requer os seguintes recursos:

- Um perfil do IAM não autenticado com permissões anexadas para:
  - Amazon Comprehend
  - Amazon Translate
  - Amazon Lex

Você pode criar esses recursos manualmente, mas recomendamos provisioná-los usando o AWS CloudFormation conforme descrito neste tutorial.

Criar os recursos da AWS usando o AWS CloudFormation

O AWS CloudFormation permite que você crie e provisione implantações de infraestrutura da AWS de maneira previsível e repetida. Para mais informações sobre o AWS CloudFormation, consulte o AWS CloudFormationGuia do usuário do .

Para criar a pilha AWS CloudFormation usando a AWS CLI:

1. Instale e configure a AWS CLI seguindo as instruções do Guia do usuário da AWS CLI.
2. Crie um arquivo chamado `setup.yaml` no diretório raiz da pasta do seu projeto e copie o conteúdo aqui GitHub para dentro dele.

   > **Note**
   >
   > O AWS CloudFormation modelo foi gerado usando o AWS CDK disponível aqui em GitHub. Para obter mais informações sobre o AWS CDK, consulte o Guia do desenvolvedor do AWS Cloud Development Kit (AWS CDK).

3. Execute o comando a seguir na linha de comando, substituindo *`STACK_NAME`* por um nome exclusivo para a pilha.

   > **Important**
   >
   > O nome da pilha deve ser exclusivo em uma Região da AWS e em uma conta da AWS. Você pode especificar até 128 caracteres, e números e hifens são permitidos.

   ```
   aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
   setup.yaml --capabilities CAPABILITY_IAM
   ```

   Para obter mais informações sobre os parâmetros do comando `create-stack`, consulte o Guia de referência de comandos da AWS CLI e o Guia do usuário do AWS CloudFormation.

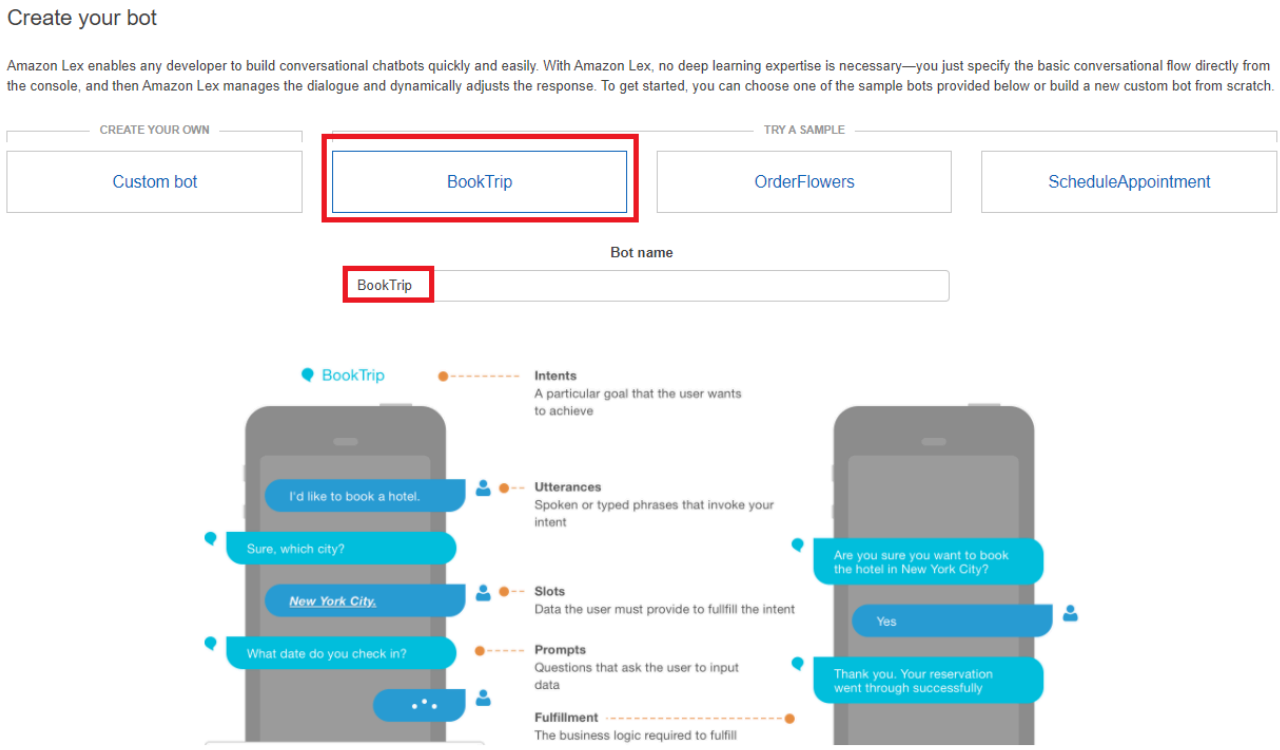
   Para visualizar os recursos criados, abra o console do Amazon Lex, escolha a pilha e selecione a guia Recursos.

Criar um bot do Amazon Lex

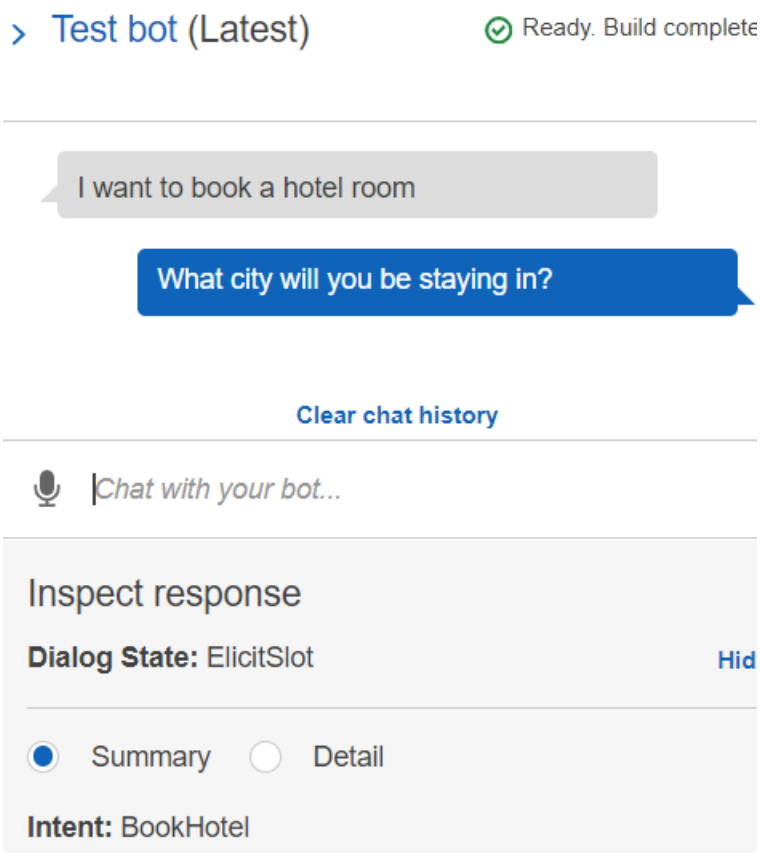
> **Important**
>
> Use a V1 do console do Amazon Lex para criar o bot. Este exemplo não funciona com bots criados usando a V2.

A primeira etapa é criar um chatbot do Amazon Lex usando o Console de Gerenciamento da Amazon Web Services. Neste exemplo, o BookTripexemplo do Amazon Lex é usado. Para obter mais informações, consulte Reservar viagem.

- Faça login no Console de Gerenciamento da Amazon Web Services e abra o console do Amazon Lex no Console da Amazon Web Services.
- Na página Bots, selecione Criar.
- Escolha o BookTripblueprint (deixe o nome padrão do bot BookTrip).



- Preencha as configurações padrão e escolha Criar (o console mostra o BookTripbot). Na guia Editor, analise os detalhes das intenções pré-configuradas.
- Teste o bot na janela de teste. Comece o teste digitando Quero reservar um quarto de hotel.



- Escolha Publicar e especifique um nome de alias (você precisará desse valor quando usar o AWS SDK for JavaScript).

> **Note**
>
> Você precisa referenciar o nome e o alias do bot em seu JavaScript código.

Criar o HTML

Crie um arquivo chamado `index.html`. Copie e cole o código abaixo em`index.html`. Esse HTML faz referência a `main.js`. Essa é uma versão empacotada do index.js, que inclui os módulos do AWS SDK for JavaScript necessários. Você criará esse arquivo em Criar o HTML. `index.html` também faz referência a `style.css`, o qual adiciona os estilos.

```
<!doctype html>
<head>
  <title>Amazon Lex - Sample Application (BookTrip)</title>
```

```
  <link type="text/css" rel="stylesheet" href="style.css" />
</head>

<body>
  <h1 id="title">Amazon Lex - BookTrip</h1>
  <p id="intro">
    This multiple language chatbot shows you how easy it is to incorporate
    <a
      href="https://aws.amazon.com/lex/"
      title="Amazon Lex (product)"
      target="_new"
      >Amazon Lex</a
    >
    into your web apps. Try it out.
  </p>
  <div id="conversation"></div>
  <input
    type="text"
    id="wisdom"
    size="80"
    value=""
    placeholder="J'ai besoin d'une chambre d'hôtel"
  />
  <br />
  <button onclick="createResponse()">Send Text</button>
  <script type="text/javascript" src="./main.js"></script>
</body>
```

Esse código também está disponível aqui em GitHub.

Criar o script do navegador

Crie um arquivo chamado `index.js`. Copie e cole o código abaixo em `index.js`. Importe os módulos e comandos do AWS SDK for JavaScript necessários. Crie clientes para o Amazon Lex, o Amazon Comprehend e o Amazon Translate. Substitua *`REGION`* por Região da AWS e *`IDENTITY_POOL_ID`* pelo ID do banco de identidades que você criou em Criar os recursos da AWS. Para recuperar esse ID do banco de identidades, abra o banco de identidades no console do Amazon Cognito, escolha Editar grupo de identidades e selecione Código de exemplo no menu lateral. O ID do banco de identidades é mostrado em vermelho no console.

Primeiro, crie um diretório `libs` e crie os objetos de cliente de serviço necessários criando três arquivos: `comprehendClient.js`, `lexClient.js` e `translateClient.js`. Cole o código apropriado abaixo em cada um e substitua *`REGION`* e *`IDENTITY_POOL_ID`* em cada arquivo.

> **Note**
>
> Use o ID do banco de identidades do Amazon Cognito que você criou em Criar os recursos da AWS usando o AWS CloudFormation.

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { ComprehendClient } from "@aws-sdk/client-comprehend";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const comprehendClient = new ComprehendClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { comprehendClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { LexRuntimeServiceClient } from "@aws-sdk/client-lex-runtime-service";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Lex service client object.
const lexClient = new LexRuntimeServiceClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
```

```
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { lexClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { TranslateClient } from "@aws-sdk/client-translate";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Translate service client object.
const translateClient = new TranslateClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { translateClient };
```

Esse código está disponível aqui em GitHub. .

Em seguida, crie um arquivo `index.js` e cole o código abaixo nele.

Substitua *`BOT_ALIAS`* e *`BOT_NAME`* pelo alias e nome do seu bot do Amazon Lex, respectivamente, e *`USER_ID`* por um ID de usuário. A função assíncrona `createResponse` faz o seguinte:

- Pega o texto inserido pelo usuário no navegador e usa o Amazon Comprehend para determinar seu código de idioma.
- Pega o código do idioma e usa o Amazon Translate para traduzir o texto para o inglês.
- Pega o texto traduzido e usa o Amazon Lex para gerar uma resposta.
- Publica a resposta na página do navegador.

```
import { DetectDominantLanguageCommand } from "@aws-sdk/client-comprehend";
```

```
import { TranslateTextCommand } from "@aws-sdk/client-translate";
import { PostTextCommand } from "@aws-sdk/client-lex-runtime-service";
import { lexClient } from "./libs/lexClient.js";
import { translateClient } from "./libs/translateClient.js";
import { comprehendClient } from "./libs/comprehendClient.js";

var g_text = "";
// Set the focus to the input box.
document.getElementById("wisdom").focus();

function showRequest() {
  var conversationDiv = document.getElementById("conversation");
  var requestPara = document.createElement("P");
  requestPara.className = "userRequest";
  requestPara.appendChild(document.createTextNode(g_text));
  conversationDiv.appendChild(requestPara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function showResponse(lexResponse) {
  var conversationDiv = document.getElementById("conversation");
  var responsePara = document.createElement("P");
  responsePara.className = "lexResponse";

  var lexTextResponse = lexResponse;

  responsePara.appendChild(document.createTextNode(lexTextResponse));
  responsePara.appendChild(document.createElement("br"));
  conversationDiv.appendChild(responsePara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function handletext(text) {
  g_text = text;
  var xhr = new XMLHttpRequest();
  xhr.addEventListener("load", loadNewItems, false);
  xhr.open("POST", "../text", true); // A Spring MVC controller
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded"); //
necessary
  xhr.send("text=" + text);
}

function loadNewItems() {
  showRequest();
```

```
  // Re-enable input.
  var wisdomText = document.getElementById("wisdom");
  wisdomText.value = "";
  wisdomText.locked = false;
}

// Respond to user's input.
const createResponse = async () => {
  // Confirm there is text to submit.
  var wisdomText = document.getElementById("wisdom");
  if (wisdomText && wisdomText.value && wisdomText.value.trim().length > 0) {
    // Disable input to show it is being sent.
    var wisdom = wisdomText.value.trim();
    wisdomText.value = "...";
    wisdomText.locked = true;
    handletext(wisdom);

    const comprehendParams = {
      Text: wisdom,
    };
    try {
      const data = await comprehendClient.send(
        new DetectDominantLanguageCommand(comprehendParams)
      );
      console.log(
        "Success. The language code is: ",
        data.Languages[0].LanguageCode
      );
      const translateParams = {
        SourceLanguageCode: data.Languages[0].LanguageCode,
        TargetLanguageCode: "en", // For example, "en" for English.
        Text: wisdom,
      };
      try {
        const data = await translateClient.send(
          new TranslateTextCommand(translateParams)
        );
        console.log("Success. Translated text: ", data.TranslatedText);
        const lexParams = {
          botName: "BookTrip",
          botAlias: "mynewalias",
          inputText: data.TranslatedText,
          userId: "chatbot", // For example, 'chatbot-demo'.
```

```
        };
        try {
          const data = await lexClient.send(new PostTextCommand(lexParams));
          console.log("Success. Response is: ", data.message);
          var msg = data.message;
          showResponse(msg);
        } catch (err) {
          console.log("Error responding to message. ", err);
        }
      } catch (err) {
        console.log("Error translating text. ", err);
      }
    } catch (err) {
      console.log("Error identifying language. ", err);
    }
  }
};
// Make the function available to the browser.
window.createResponse = createResponse;
```

Esse código está disponível aqui em GitHub. .

Agora, use o webpack para empacotar os módulos `index.js` e AWS SDK for JavaScript em um único arquivo, `main.js`.

1. Caso ainda não tenha feito isso, siga as instruções descritas em Pré-requisitos para este exemplo para instalar o webpack.

   > **Note**
   >
   > Para obter informações sobre webpack, consulte Agrupe aplicativos com o webpack.

2. Execute o seguinte na linha de comando para agrupar o deste JavaScript exemplo em um arquivo chamado`main.js`:

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

Próximas etapas

Parabéns! Você criou um aplicativo Node.js que usa o Amazon Lex para criar uma experiência de usuário interativa. Conforme informado no início deste tutorial, certifique-se de encerrar todos os

recursos que criar enquanto percorre este tutorial para garantir que você não seja cobrado. Você pode fazer isso excluindo a pilha AWS CloudFormation que criou no tópico Criar os recursos da AWS deste tutorial, da seguinte forma:

1. Abra o console de AWS CloudFormation.
2. Na página Pilhas, selecione a pilha.
3. Escolha Excluir.

Para obter mais exemplos entre serviços da AWS, consulte Exemplos entre serviços da AWS SDK for JavaScript.

Criação de um exemplo de aplicativo de mensagens

Você pode criar um aplicativo da AWS que envia e recupera mensagens usando o AWS SDK for JavaScript e o Amazon Simple Queue Service (Amazon SQS). As mensagens são armazenadas em uma fila FIFO (primeiro a entrar, primeiro a sair) que garante que a ordem das mensagens seja consistente. Por exemplo, a primeira mensagem armazenada na fila é a primeira mensagem lida da fila.

> **Note**
>
> Para obter mais informações sobre o Amazon SQS, consulte O que é o Amazon Simple Queue Service?

Neste tutorial, você cria um aplicativo Node.js chamado AWS Messaging.

Custo para concluir: os serviços da AWS incluídos neste documento estão incluídos no Nível gratuito da AWS.

Observação: certifique-se de encerrar todos os recursos que você cria ao passar por este tutorial para garantir que você não seja cobrado.

Para criar o aplicativo:

1. Pré-requisitos
2. Provisionar recursos
3. Entender o fluxo de trabalho

4. Criar o HTML
5. Criar o script do navegador
6. Próximas etapas

Pré-requisitos

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Configure o ambiente do projeto para executar esses TypeScript exemplos de Node e instale os módulos necessários AWS SDK for JavaScript e de terceiros. Siga as instruções em GitHub.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhado, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de ferramentas e SDKs da AWS.

> **Important**
>
> Este exemplo usa ECMAScript6 (ES6). Isso requer o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js.
> No entanto, se você preferir usar a sintaxe CommonJS, consulte Sintaxe ES6/CommonJS de JavaScript.

Criar os recursos da AWS

Este tutorial requer os seguintes recursos:

- Um perfil do IAM não autenticado com permissões para Amazon SQS.
- Uma fila FIFO do Amazon SQS chamada Message.fifo. Para obter informações sobre a criação de uma fila, consulte Criação de uma fila do Amazon SQS.

Você pode criar esses recursos manualmente, mas recomendamos provisioná-los usando o AWS CloudFormation (AWS CloudFormation) conforme descrito neste tutorial.

> **Note**
>
> O AWS CloudFormation é uma estrutura de desenvolvimento de software que permite definir recursos de aplicativos de nuvem. Para obter mais informações, consulte o AWS CloudFormationGuia do Usuário.

Criar os recursos da AWS usando o AWS CloudFormation

O AWS CloudFormation permite que você crie e provisione implantações de infraestrutura da AWS de maneira previsível e repetida. Para mais informações sobre o AWS CloudFormation, consulte o AWS CloudFormationGuia do usuário do .

Para criar a pilha AWS CloudFormation usando a AWS CLI:

1. Instale e configure a AWS CLI seguindo as instruções do Guia do usuário da AWS CLI.
2. Crie um arquivo chamado `setup.yaml` no diretório raiz da pasta do seu projeto e copie o conteúdo aqui GitHub para dentro dele.

   > **Note**
   >
   > O AWS CloudFormation modelo foi gerado usando o AWS CDK disponível aqui em GitHub. Para obter mais informações sobre o AWS CDK, consulte o Guia do desenvolvedor do AWS Cloud Development Kit (AWS CDK).

3. Execute o comando a seguir na linha de comando, substituindo *`STACK_NAME`* por um nome exclusivo para a pilha.

   > **Important**
   >
   > O nome da pilha deve ser exclusivo em uma Região da AWS e em uma conta da AWS. Você pode especificar até 128 caracteres, e números e hifens são permitidos.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

Para obter mais informações sobre os parâmetros do comando `create-stack`, consulte o Guia de referência de comandos da AWS CLI e o Guia do usuário do AWS CloudFormation.

Para visualizar os recursos criados, abra o AWS CloudFormation no console de gerenciamento do AWS, escolha a pilha e selecione a guia Recursos.

Entendar o aplicativo AWS Messaging

Para enviar uma mensagem para uma fila do SQS, insira a mensagem no aplicativo e selecione Enviar.

Depois que a mensagem é enviada, o aplicativo exibe a mensagem.

Você pode escolher Depurar para depurar as mensagens da fila do Amazon SQS. Isso resulta em uma fila vazia e nenhuma mensagem é exibida no aplicativo.

As informações a seguir descrevem como o aplicativo lida com uma mensagem:

- O usuário seleciona seu nome, insere uma mensagem e envia a mensagem, o que inicia a função `pushMessage`.
- `pushMessage` recupera o URL de fila do Amazon SQS e envia uma mensagem com um valor de ID de mensagem exclusivo (um GUID), o texto da mensagem e o usuário para a fila do Amazon SQS.
- `pushMessage` recupera as mensagens da Fila do Amazon SQS, extrai o usuário e a mensagem de cada mensagem e exibe as mensagens.
- O usuário pode limpar as mensagens, o que exclui as mensagens da fila do Amazon SQS e da interface do usuário.

Criar a página HTML

Agora você cria os arquivos HTML necessários para a interface gráfica do usuário (GUI) do aplicativo. Crie um arquivo chamado `index.html`. Copie e cole o código abaixo em`index.html`. Esse HTML faz referência a `main.js`. Essa é uma versão empacotada do index.js, que inclui os módulos do AWS SDK for JavaScript necessários.

```
<!doctype html>
<html
  xmlns:th="http://www.thymeleaf.org"
```

```
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" href="./images/favicon.ico" />
    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    />
    <link rel="stylesheet" href="./css/styles.css" />
    <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
    <script src="https://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
    <script src="./js/main.js"></script>
    <style>
      .messageelement {
        margin: auto;
        border: 2px solid #dedede;
        background-color: #d7d1d0;
        border-radius: 5px;
        max-width: 800px;
        padding: 10px;
        margin: 10px 0;
      }

      .messageelement::after {
        content: "";
        clear: both;
        display: table;
      }

      .messageelement img {
        float: left;
        max-width: 60px;
        width: 100%;
        margin-right: 20px;
        border-radius: 50%;
      }

      .messageelement img.right {
        float: right;
        margin-left: 20px;
        margin-right: 0;
```

```
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h2>AWS Sample Messaging Application</h2>
      <div id="messages"></div>

      <div class="input-group mb-3">
        <div class="input-group-prepend">
          <span class="input-group-text" id="basic-addon1">Sender:</span>
        </div>
        <select name="cars" id="username">
          <option value="Scott">Brian</option>
          <option value="Tricia">Tricia</option>
        </select>
      </div>

      <div class="input-group">
        <div class="input-group-prepend">
          <span class="input-group-text">Message:</span>
        </div>
        <textarea
          class="form-control"
          id="textarea"
          aria-label="With textarea"
        ></textarea>
        <button
          type="button"
          onclick="pushMessage()"
          id="send"
          class="btn btn-success"
        >
          Send
        </button>
        <button
          type="button"
          onclick="purge()"
          id="refresh"
          class="btn btn-success"
        >
          Purge
        </button>
      </div>
```

```
      <!-- All of these child items are hidden and only displayed in a FancyBox
 ------------------------------------------------------>
      <div id="hide" style="display: none">
        <div id="base" class="messageelement">
          <img
            src="../public/images/av2.png"
            alt="Avatar"
            class="right"
            style="width: 100%"
          />
          <p id="text">Excellent! So, what do you want to do today?</p>
          <span class="time-right">11:02</span>
        </div>
      </div>
    </div>
  </body>
</html>
```

Esse código também está disponível aqui em GitHub.

Criação do script do navegador

Neste tópico, você cria um script do navegador para o aplicativo. Depois de criar o script do navegador, você o empacota em um arquivo chamado `main.js`, conforme descrito em Agrupando o JavaScript.

Crie um arquivo chamado `index.js`. Copie e cole o código daqui em diante GitHub nele.

Esse código é explicado na seção a seguir:

1. Configuração
2. populateChat
3. pushmessages
4. purge

Configuração

Primeiro, crie um diretório `libs` e crie o objeto de cliente do Amazon SQS necessário criando um arquivo denominado `sqsClient.js`. Substitua *`REGION`* e *`IDENTITY_POOL_ID`* em cada uma.

> **Note**
>
> Use o ID do banco de identidades do Amazon Cognito que você criou em Criar os recursos da AWS.

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import {SQSClient} from "@aws-sdk/client-sqs";
const REGION = "REGION"; //e.g. "us-east-1"
const IdentityPoolId = "IDENTITY_POOL_ID";
const sqsClient = new SQSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IdentityPoolId
  }),
});
```

No `index.js`, importe os módulos e comandos do AWS SDK for JavaScript necessários. Substitua *`SQS_QUEUE_ARN`* pelo nome da fila do Amazon SQS; que você criou em Criar os recursos da AWS.

```
import {
  GetQueueUrlCommand,
  SendMessageCommand,
  ReceiveMessageCommand,
  PurgeQueueCommand,
} from "@aws-sdk/client-sqs";
import { sqsClient } from "./libs/sqsClient.js";

const QueueName = "SQS_QUEUE_NAME"; // The Amazon SQS queue name, which must end
 in .fifo for this example.
```

populateChat

A função onload `populateChat` recupera automaticamente o URL da fila do Amazon SQS, recupera todas as mensagens que estão na fila e as exibe.

```
$(function () {
```

```
  populateChat();
});

const populateChat = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for retrieving the messages in the Amazon SQS Queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };
    try {
      // Retrieve the messages from the Amazon SQS Queue.
      const data = await sqsClient.send(
        new ReceiveMessageCommand(getMessageParams)
      );
      console.log("Successfully retrieved messages", data.Messages);

      // Loop through messages for user and message body.
      var i;
      for (i = 0; i < data.Messages.length; i++) {
        const name = data.Messages[i].MessageAttributes.Name.StringValue;
        const body = data.Messages[i].Body;
        // Create the HTML for the message.
        var userText = body + "<br><br><b>" + name;
        var myTextNode = $("#base").clone();
        myTextNode.text(userText);
        var image_url;
        var n = name.localeCompare("Scott");
        if (n == 0) image_url = "./images/av1.png";
        else image_url = "./images/av2.png";
```

```
        var images_div =
          '<img src="' +
          image_url +
          '" alt="Avatar" class="right" style=""width:100%;"">';
        myTextNode.html(userText);
        myTextNode.append(images_div);

        // Add the message to the GUI.
        $("#messages").append(myTextNode);
      }
    } catch (err) {
      console.log("Error loading messages: ", err);
    }
  } catch (err) {
    console.log("Error retrieving SQS queue URL: ", err);
  }
};
```

Enviar mensagens por push

O usuário seleciona seu nome, insere a mensagem e a envia, iniciando a função `pushMessage`. `pushMessage` recupera o URL da fila do Amazon SQS e, em seguida, envia uma mensagem com um valor de ID de mensagem exclusivo (um GUID), o texto da mensagem e o usuário para a fila do Amazon SQS. Em seguida, ele recupera todas as mensagens da fila do Amazon SQS e as exibe.

```
const pushMessage = async () => {
  // Get and convert user and message input.
  var user = $("#username").val();
  var message = $("#textarea").val();

  // Create random deduplication ID.
  var dt = new Date().getTime();
  var uuid = "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (
    c
  ) {
    var r = (dt + Math.random() * 16) % 16 | 0;
    dt = Math.floor(dt / 16);
    return (c == "x" ? r : (r & 0x3) | 0x8).toString(16);
  });

  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
```

```
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for the message.
    var messageParams = {
      MessageAttributes: {
        Name: {
          DataType: "String",
          StringValue: user,
        },
      },
      MessageBody: message,
      MessageDeduplicationId: uuid,
      MessageGroupId: "GroupA",
      QueueUrl: data.QueueUrl,
    };
    const result = await sqsClient.send(new SendMessageCommand(messageParams));
    console.log("Success", result.MessageId);

    // Set the parameters for retrieving all messages in the SQS queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };

    // Retrieve messages from SQS Queue.
    const final = await sqsClient.send(
      new ReceiveMessageCommand(getMessageParams)
    );
    console.log("Successfully retrieved", final.Messages);
    $("#messages").empty();
    // Loop through messages for user and message body.
    var i;
    for (i = 0; i < final.Messages.length; i++) {
      const name = final.Messages[i].MessageAttributes.Name.StringValue;
      const body = final.Messages[i].Body;
```

```
      // Create the HTML for the message.
      var userText = body + "<br><br><b>" + name;
      var myTextNode = $("#base").clone();
      myTextNode.text(userText);
      var image_url;
      var n = name.localeCompare("Scott");
      if (n == 0) image_url = "./images/av1.png";
      else image_url = "./images/av2.png";
      var images_div =
        '<img src="' +
        image_url +
        '" alt="Avatar" class="right" style=""width:100%;"">';
      myTextNode.html(userText);
      myTextNode.append(images_div);
      // Add the HTML to the GUI.
      $("#messages").append(myTextNode);
    }
  } catch (err) {
    console.log("Error", err);
  }
};
// Make the function available to the browser window.
window.pushMessage = pushMessage;
```

Eliminar mensagens.

purge exclui as mensagens da fila do Amazon SQS e da interface do usuário.

```
// Delete the message from the Amazon SQS queue.
const purge = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success", data.QueueUrl);
    // Delete all the messages in the Amazon SQS Queue.
    const result = await sqsClient.send(
```

```
      new PurgeQueueCommand({ QueueUrl: data.QueueUrl })
    );
    // Delete all the messages from the GUI.
    $("#messages").empty();
    console.log("Success. All messages deleted.", data);
  } catch (err) {
    console.log("Error", err);
  }
};

// Make the function available to the browser window.
window.purge = purge;
```

Agrupando o JavaScript

Esse código completo do script do navegador está disponível aqui. GitHub .

Agora, use o webpack para empacotar os módulos `index.js` e AWS SDK for JavaScript em um único arquivo, `main.js`.

1. Caso ainda não tenha feito isso, siga as instruções descritas em Pré-requisitos para este exemplo para instalar o webpack.

   > **Note**
   >
   > Para obter informações sobre webpack, consulte Agrupe aplicativos com o webpack.

2. Execute o seguinte na linha de comando para agrupar o deste JavaScript exemplo em um arquivo chamado`<index.js>`:

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

Próximas etapas

Parabéns! Você criou e implantou o aplicativo AWS Messaging que usa o Amazon SQS. Conforme informado no início deste tutorial, certifique-se de encerrar todos os recursos que criar enquanto percorre este tutorial para garantir que você não seja mais cobrado. Você pode fazer isso excluindo a pilha AWS CloudFormation que criou no tópico Criar os recursos da AWS deste tutorial, da seguinte forma:

1. Abra o AWS CloudFormation no console de gerenciamento da AWS.
2. Abra a página Pilhas e selecione a pilha.
3. Escolha Excluir.

Use AWS Cloud9 com o AWS SDK for JavaScript

Você pode usar AWS Cloud9 com o AWS SDK for JavaScript para escrever e executar seu código JavaScript no navegador, bem como escrever, executar e depurar seu código Node.js, usando apenas um navegador. AWS Cloud9 inclui ferramentas como um editor de código e um terminal, além de um depurador para o código Node.js.

Como o AWS Cloud9 IDE é baseado em nuvem, você pode trabalhar em seus projetos no escritório, em casa ou em qualquer lugar usando uma máquina conectada à Internet. Para obter informações gerais sobre AWS Cloud9, consulte o Guia AWS Cloud9 do usuário.

As etapas a seguir descrevem como configurar AWS Cloud9 o SDK para JavaScript.

Sumário

Etapa 1: configurar sua AWS conta para usar AWS Cloud9

Comece a usar AWS Cloud9 fazendo login no AWS Cloud9 console como uma entidade AWS Identity and Access Management (IAM) (por exemplo, um usuário do IAM) que tem permissões de acesso AWS Cloud9 em sua AWS conta.

Para configurar uma entidade do IAM em sua AWS conta para acessar AWS Cloud9 e fazer login no AWS Cloud9 console, consulte Configuração da equipe AWS Cloud9 no Guia do AWS Cloud9 usuário.

Etapa 2: configurar seu ambiente AWS Cloud9 de desenvolvimento

Depois de entrar no AWS Cloud9 console, use o console para criar um ambiente de AWS Cloud9 desenvolvimento. Depois de criar o ambiente, AWS Cloud9 abre o IDE para esse ambiente.

Para obter detalhes, consulte Criação de um ambiente no AWS Cloud9 no Guia do usuário do AWS Cloud9 .

> **Note**
>
> Ao criar o ambiente no console pela primeira vez, recomendamos selecionar a opção Create a new instance for environment (EC2) (Criar um nova instância para o ambiente (EC2)). Essa opção diz AWS Cloud9 para criar um ambiente, iniciar uma instância do Amazon EC2 e, em seguida, conectar a nova instância ao novo ambiente. Essa é a maneira mais rápida de começar a usar AWS Cloud9.

Etapa 3: configurar o SDK para JavaScript

Depois de AWS Cloud9 abrir o IDE para seu ambiente de desenvolvimento, siga um ou ambos os procedimentos a seguir para usar o IDE para configurar o SDK JavaScript em seu ambiente.

Para configurar o SDK JavaScript para Node.js

1. Se o terminal ainda não estiver aberto no IDE, abra-o. Para isso, na barra de menus no IDE, escolha Window, New Terminal (Janela, novo terminal).
2. Execute o comando a seguir `npm` para usar na instalação do `Cloud9` cliente do SDK para JavaScript.

   ```
   npm install @aws-sdk/client-cloud9
   ```

   Caso o IDE não consiga encontrar `npm`, execute os comandos a seguir, um por vez, na ordem a seguir, para instalar `npm`. (Esses comandos pressupõem que você tenha escolhido a opção Create a new instance for environment (EC2) (Criar uma nova instância para o ambiente [EC2]), anteriormente neste tópico.)

   > **Warning**
   >
   > AWS não controla o código a seguir. Antes de executar, certifique-se de verificar sua autenticidade e integridade. Mais informações sobre esse código podem ser encontradas no repositório nvm (Node Version Manager) GitHub .

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash #
 Download and install Node Version Manager (nvm).
. ~/.bashrc                                                                    #
 Activate nvm.
nvm install node                                                               #
 Use nvm to install npm (and Node.js at the same time).
```

Para configurar o SDK JavaScript no navegador

Para usar o SDK JavaScript em suas páginas HTML, use WebPack para agrupar os módulos de cliente necessários e todas as JavaScript funções necessárias em um único JavaScript arquivo e adicioná-lo em uma tag <head> de script em suas páginas HTML. Por exemplo: .

```
<script src=./main.js></script>
```

> **Note**
>
> Para obter mais informações sobre o WebPack, consulte Agrupe aplicativos com o webpack.

Etapa 4: Fazer download do código de exemplo

Use o terminal que você abriu na etapa anterior para baixar o código de exemplo do SDK JavaScript para o ambiente de AWS Cloud9 desenvolvimento. (Se o terminal ainda não estiver aberto no IDE, abra-o escolhendo Window, New Terminal (Janela, novo terminal) na barra de menus no IDE.)

Para fazer download do código de exemplo, execute o comando a seguir. Esse comando baixa uma cópia de todos os exemplos de código usados na documentação oficial do AWS SDK no diretório raiz do seu ambiente.

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

Para encontrar exemplos de código para o SDK JavaScript, use a janela Ambiente para abrir o*ENVIRONMENT_NAME*\aws-doc-sdk-examples\javascriptv3\example_code/src, onde *ENVIRONMENT_NAME é o nome do seu ambiente* de desenvolvimento. AWS Cloud9

Para saber como trabalhar com esses e outros exemplos de código, consulte SDK para ver exemplos de JavaScript código.

Etapa 5: Executar e depurar o código de exemplo

Para executar código em seu ambiente de AWS Cloud9 desenvolvimento, consulte Executar seu código no Guia AWS Cloud9 do usuário.

Para depurar código Node.js, consulte Depurar o código no Guia do usuário do AWS Cloud9 .

SDK para exemplos de JavaScript código (v3)

Os exemplos de código neste tópico mostram como usar o AWS SDK for JavaScript (v3) com 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.

Exemplos

- Ações e cenários usando o SDK para JavaScript (v3)
- Exemplos de serviços cruzados usando o SDK para JavaScript (v3)

Ações e cenários usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com Serviços 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.

Serviços

- Exemplos de Auto Scaling usando SDK para JavaScript (v3)
- Exemplos do Amazon Bedrock usando o SDK para JavaScript (v3)
- Exemplos de tempo de execução do Amazon Bedrock usando SDK para JavaScript (v3)
- Exemplos de agentes do Amazon Bedrock usando SDK para JavaScript (v3)
- Exemplos de agentes para Amazon Bedrock Runtime usando SDK para JavaScript (v3)

- CloudWatch exemplos usando o SDK para JavaScript (v3)
- CloudWatch Exemplos de eventos usando o SDK para JavaScript (v3)
- CloudWatch Exemplos de registros usando o SDK para JavaScript (v3)
- CodeBuild exemplos usando o SDK para JavaScript (v3)
- Exemplos de provedores de identidade Amazon Cognito usando SDK para JavaScript (v3)
- Exemplos do Amazon DocumentDB usando SDK para JavaScript (v3)
- Exemplos do DynamoDB usando o SDK JavaScript para (v3)
- Exemplos do Amazon EC2 usando SDK para JavaScript (v3)
- Elastic Load Balancing — Exemplos da versão 2 usando SDK para JavaScript (v3)
- EventBridge exemplos usando o SDK para JavaScript (v3)
- AWS Glue exemplos usando o SDK para JavaScript (v3)
- HealthImaging exemplos usando o SDK para JavaScript (v3)
- Exemplos de IAM usando SDK para JavaScript (v3)
- Exemplos do Kinesis usando o SDK para JavaScript (v3)
- Exemplos de Lambda usando SDK para JavaScript (v3)
- Exemplos do Amazon Personalize usando o SDK para JavaScript (v3)
- Exemplos de eventos do Amazon Personalize usando SDK para JavaScript (v3)
- Exemplos do Amazon Personalize Runtime usando SDK para JavaScript (v3)
- Exemplos do Amazon Pinpoint usando SDK para JavaScript (v3)
- Exemplos do Amazon Redshift usando SDK para JavaScript (v3)
- Exemplos do Amazon S3 usando SDK para JavaScript (v3)
- Exemplos do S3 Glacier usando SDK para JavaScript (v3)
- SageMaker exemplos usando o SDK para JavaScript (v3)
- Exemplos de Secrets Manager usando SDK para JavaScript (v3)
- Exemplos do Amazon SES usando SDK para JavaScript (v3)
- Exemplos do Amazon SNS usando SDK para JavaScript (v3)
- Exemplos do Amazon SQS usando SDK para JavaScript (v3)
- Exemplos de Step Functions usando o SDK para JavaScript (v3)
- AWS STS exemplos usando o SDK para JavaScript (v3)

- AWS Support exemplos usando o SDK para JavaScript (v3)
- Exemplos do Amazon Transcribe usando SDK JavaScript para (v3)

Exemplos de Auto Scaling usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com Auto Scaling.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações
- Cenários

Ações

`AttachLoadBalancerTargetGroups`

O código de exemplo a seguir mostra como usar `AttachLoadBalancerTargetGroups`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const client = new AutoScalingClient({});
```

```
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
```

- Para obter detalhes da API, consulte AttachLoadBalancerTargetGrupos na Referência AWS SDK for JavaScript da API.

Cenários

Criar e gerenciar um serviço resiliente

O exemplo de código a seguir mostra como criar um serviço web com balanceamento de carga que retorna recomendações de livros, filmes e músicas. O exemplo mostra como o serviço responde a falhas e como é possível reestruturá-lo para gerar mais resiliência em caso de falhas.

- Use um grupo do Amazon EC2 Auto Scaling para criar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) com base em um modelo de execução e para manter o número de instâncias em um intervalo especificado.
- Gerencie e distribua solicitações HTTP com o Elastic Load Balancing.
- Monitore a integridade das instâncias em um grupo do Auto Scaling e encaminhe solicitações somente para instâncias íntegras.
- Execute um servidor Web Python em cada instância do EC2 para lidar com solicitações HTTP. O servidor Web responde com recomendações e verificações de integridade.
- Simule um serviço de recomendação com uma tabela do Amazon DynamoDB.
- Controle a resposta do servidor web às solicitações e verificações de saúde atualizando AWS Systems Manager os parâmetros.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Execute o cenário interativo em um prompt de comando.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};
```

```
// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

Criar etapas para implantar todos os recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
```

```
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
```

```
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );
```

```
    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
```

```
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
```

```
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
```

```
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
```

```
    }),
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({
        AvailabilityZones: state.availabilityZoneNames,
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
          LaunchTemplateName: NAMES.launchTemplateName,
          Version: "$Default",
        },
        MinSize: 3,
        MaxSize: 3,
      }),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
```

```
    */
  (state) =>
    MESSAGES.createdAutoScalingGroup
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
      .replace(
        "${AVAILABILITY_ZONE_NAMES}",
        state.availabilityZoneNames.join(", "),
      ),
),
new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
  type: "confirm",
}),
new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
new ScenarioAction("getVpc", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
```

```
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
```

```
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
```

```
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
```

```
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
```

```
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
```

```
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

Criar etapas para executar a demonstração.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
```

```
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
```

```
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
```

```
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];
```

```
/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
```

```
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
```

```
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );
```

```
        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
```

```
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
```

```
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
```

```
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
```

```
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Criar etapas para destruir todos os recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
```

```
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
```

```
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
```

```
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
```

```
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
```

```
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
```

```
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
```

```
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
```

```
  // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );

    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      client.send(
        new DeleteTargetGroupCommand({
          TargetGroupArn: TargetGroups[0].TargetGroupArn,
        }),
      ),
    );
  } catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
  }
  // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  } else {
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
```

```
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
```

```
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
```

```
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
```

```
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGrupos
  - CreateAutoScalingGroup
  - CreateInstancePerfil
  - CreateLaunchModelo
  - CreateListener
  - CreateLoadBalanceador
  - CreateTargetGrupo
  - DeleteAutoScalingGroup
  - DeleteInstancePerfil
  - DeleteLaunchModelo
  - DeleteLoadBalanceador
  - DeleteTargetGrupo
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZonas
  - DescribeIamInstanceProfileAssociações
  - DescribeInstances
  - DescribeLoadBalanceadores
  - DescribeSubnets
  - DescribeTargetGrupos
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociação
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Exemplos do Amazon Bedrock usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Bedrock.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Amazon Bedrock

Os exemplos de código a seguir mostram por onde começar a usar o Amazon Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

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

import { fileURLToPath } from "url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

const REGION = "us-east-1";
const client = new BedrockClient({ region: REGION });
```

```
export const main = async () => {
  const command = new ListFoundationModelsCommand({});

  const response = await client.send(command);
  const models = response.modelSummaries;

  console.log("Listing the available Bedrock foundation models:");

  for (let model of models) {
    console.log("=".repeat(42));
    console.log(` Model: ${model.modelId}`);
    console.log("-".repeat(42));
    console.log(` Name: ${model.modelName}`);
    console.log(` Provider: ${model.providerName}`);
    console.log(` Model ARN: ${model.modelArn}`);
    console.log(` Input modalities: ${model.inputModalities}`);
    console.log(` Output modalities: ${model.outputModalities}`);
    console.log(` Supported customizations: ${model.customizationsSupported}`);
    console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
    console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
    console.log("=".repeat(42) + "\n");
  }

  const active = models.filter(
    (m) => m.modelLifecycle.status === "ACTIVE",
  ).length;
  const legacy = models.filter(
    (m) => m.modelLifecycle.status === "LEGACY",
  ).length;

  console.log(
    `There are ${active} active and ${legacy} legacy foundation models in
 ${REGION}.`,
  );

  return response;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- Para obter detalhes da API, consulte ListFoundationModelos na Referência AWS SDK for JavaScript da API.

Tópicos

Ações

`GetFoundationModel`

O código de exemplo a seguir mostra como usar `GetFoundationModel`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha detalhes de um modelo de base.

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

import { fileURLToPath } from "url";

import {
  BedrockClient,
  GetFoundationModelCommand,
} from "@aws-sdk/client-bedrock";

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 models.
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();
```

```
  const command = new GetFoundationModelCommand({
    modelIdentifier: "amazon.titan-embed-text-v1",
  });

  const response = await client.send(command);

  return response.modelDetails;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const model = await getFoundationModel();
  console.log(model);
}
```

- Para obter detalhes da API, consulte GetFoundationModelo na Referência AWS SDK for JavaScript da API.

`ListFoundationModels`

O código de exemplo a seguir mostra como usar `ListFoundationModels`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os modelos de base disponíveis.

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

import { fileURLToPath } from "url";

import {
  BedrockClient,
```

```
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

/**
 * List the available Amazon Bedrock foundation models.
 *
 * @return {FoundationModelSummary[]} - The list of available bedrock foundation
 models.
 */
export const listFoundationModels = async () => {
  const client = new BedrockClient();

  const input = {
    // byProvider: 'STRING_VALUE',
    // byCustomizationType: 'FINE_TUNING' || 'CONTINUED_PRE_TRAINING',
    // byOutputModality: 'TEXT' || 'IMAGE' || 'EMBEDDING',
    // byInferenceType: 'ON_DEMAND' || 'PROVISIONED',
  };

  const command = new ListFoundationModelsCommand(input);

  const response = await client.send(command);

  return response.modelSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const models = await listFoundationModels();
  console.log(models);
}
```

- Para obter detalhes da API, consulte ListFoundationModelos na Referência AWS SDK for JavaScript da API.

Exemplos de tempo de execução do Amazon Bedrock usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Bedrock Runtime.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Amazon Bedrock

Os exemplos de código a seguir mostram por onde começar a usar o Amazon Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

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

/**
 * @typedef {Object} Content
 * @property {string} text
 *
 * @typedef {Object} Usage
 * @property {number} input_tokens
 * @property {number} output_tokens
 *
 * @typedef {Object} ResponseBody
 * @property {Content[]} content
 * @property {Usage} usage
 */

import { fileURLToPath } from "url";
import {
```

```
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

const AWS_REGION = "us-east-1";

const MODEL_ID = "anthropic.claude-3-haiku-20240307-v1:0";
const PROMPT = "Hi. In a short paragraph, explain what you can do.";

const hello = async () => {
  console.log("=".repeat(35));
  console.log("Welcome to the Amazon Bedrock demo!");
  console.log("=".repeat(35));

  console.log("Model: Anthropic Claude 3 Haiku");
  console.log(`Prompt: ${PROMPT}\n`);
  console.log("Invoking model...\n");

  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: AWS_REGION });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [{ role: "user", content: [{ type: "text", text: PROMPT }] }],
  };

  // Invoke Claude with the payload and wait for the response.
  const apiResponse = await client.send(
    new InvokeModelCommand({
      contentType: "application/json",
      body: JSON.stringify(payload),
      modelId: MODEL_ID,
    }),
  );

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  const responses = responseBody.content;

  if (responses.length === 1) {
```

```
    console.log(`Response: ${responses[0].text}`);
  } else {
    console.log("Haiku returned multiple responses:");
    console.log(responses);
  }

  console.log(`\nNumber of input tokens:   ${responseBody.usage.input_tokens}`);
  console.log(`Number of output tokens: ${responseBody.usage.output_tokens}`);
};

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await hello();
}
```

- Para obter detalhes da API, consulte InvokeModela Referência AWS SDK for JavaScript da API.

Tópicos

- AI21 Labs Jurassic-2
- Texto Amazon Titan
- Anthropic Claude
- Cohere Command
- Lhama de metal
- IA Mistral
- Cenários

AI21 Labs Jurassic-2

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o AI21 Labs Jurassic-2, usando a API Converse do Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para o AI21 Labs Jurassic-2, usando a API Converse do Bedrock.

```
// Use the Conversation API to send a text message to AI21 Labs Jurassic-2.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Jurassic-2 Mid.
const modelId = "ai21.j2-mid-v1";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
```

```
  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte Converse em Referência de AWS SDK for JavaScript API.

InvokeModel

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o AI21 Labs Jurassic-2, usando a API Invoke Model.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto.

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Data
 * @property {string} text
```

```
 *
 * @typedef {Object} Completion
 * @property {Data} data
 *
 * @typedef {Object} ResponseBody
 * @property {Completion[]} completions
 */

/**
 * Invokes an AI21 Labs Jurassic-2 model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "ai21.j2-mid-
v1".
 */
export const invokeModel = async (prompt, modelId = "ai21.j2-mid-v1") => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    prompt,
    maxTokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s).
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.completions[0].data.text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
```

```
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.JURASSIC2_MID.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- Para obter detalhes da API, consulte InvokeModela Referência AWS SDK for JavaScript da API.

Texto Amazon Titan

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Amazon Titan Text usando a API Converse do Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para o Amazon Titan Text usando a API Converse da Bedrock.

```
// Use the Conversation API to send a text message to Amazon Titan Text.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";
```

```
// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Titan Text Premier.
const modelId = "amazon.titan-text-premier-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte Converse em Referência de AWS SDK for JavaScript API.

ConverseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Amazon Titan Text usando a API Converse da Bedrock e processar o fluxo de resposta em tempo real.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para o Amazon Titan Text usando a API Converse da Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Conversation API to send a text message to Amazon Titan Text.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Titan Text Premier.
const modelId = "amazon.titan-text-premier-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
```

```
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK for JavaScript da API.

InvokeModel

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Amazon Titan Text usando a API Invoke Model.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto.

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
```

```
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {Object[]} results
 */

/**
 * Invokes an Amazon Titan Text generation model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "amazon.titan-text-express-v1".
 */
export const invokeModel = async (
  prompt,
  modelId = "amazon.titan-text-express-v1",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    inputText: prompt,
    textGenerationConfig: {
      maxTokenCount: 4096,
      stopSequences: [],
      temperature: 0,
      topP: 1,
    },
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
```

```
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.results[0].outputText;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.TITAN_TEXT_G1_EXPRESS.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- Para obter detalhes da API, consulte InvokeModela Referência AWS SDK for JavaScript da API.

Anthropic Claude

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto para Anthropic Claude usando a API Converse do Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para Anthropic Claude usando a API Converse do Bedrock.

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte Converse em Referência de AWS SDK for JavaScript API.

ConverseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto para Anthropic Claude usando a API Converse da Bedrock e processar o fluxo de resposta em tempo real.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para Anthropic Claude usando a API Converse da Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
```

```
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK for JavaScript da API.

InvokeModel

O exemplo de código a seguir mostra como enviar uma mensagem de texto para Anthropic Claude usando a API Invoke Model.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
```

```
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
```

```
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;

    if (chunk_type === "content_block_delta") {
      const text = chunk.delta.text;
      completeMessage = completeMessage + text;
      process.stdout.write(text);
    }
  }

  // Return the final response
  return completeMessage;
};
```

```
// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log("\n" + "-".repeat(53));
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- Para obter detalhes da API, consulte InvokeModela Referência AWS SDK for JavaScript da API.

InvokeModelWithResponseTransmitir

O exemplo de código a seguir mostra como enviar uma mensagem de texto para modelos da Anthropic Claude, usando a API Invoke Model, e imprimir o fluxo de resposta.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto e processar o fluxo de resposta em tempo real.

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

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
```

```
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
```

```
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;

    if (chunk_type === "content_block_delta") {
      const text = chunk.delta.text;
      completeMessage = completeMessage + text;
      process.stdout.write(text);
    }
  }

  // Return the final response
  return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
```

```
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log("\n" + "-".repeat(53));
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- Para obter detalhes da API, consulte InvokeModelWithResponseStream na Referência AWS SDK for JavaScript da API.

Cohere Command

Converse: Todos os modelos

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Comando Cohere, usando a API Converse da Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para o Cohere Command, usando a API Converse do Bedrock.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
```

```
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte Converse em Referência de AWS SDK for JavaScript API.

ConverseStream: Todos os modelos

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Comando Cohere usando a API Converse da Bedrock e processar o fluxo de resposta em tempo real.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para o Cohere Command usando a API Converse da Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
```

```
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK for JavaScript da API.

Lhama de metal

Todos os modelos: Converse API

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Meta Llama usando a API Converse do Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para Meta Llama usando a API Converse do Bedrock.

```
// Use the Conversation API to send a text message to Meta Llama.
```

```
import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte Converse em Referência de AWS SDK for JavaScript API.

ConverseStream: Todos os modelos

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Meta Llama usando a API Converse da Bedrock e processar o fluxo de resposta em tempo real.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para o Meta Llama usando a API Converse da Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
```

```
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK for JavaScript da API.

InvokeModel: Lhama 2

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Meta Llama 2 usando a API Invoke Model.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto.

```
// Send a prompt to Meta Llama 2 and print the response.

import {
```

```
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 2 Chat 13B.
const modelId = "meta.llama2-13b-chat-v1";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 2's prompt format.
const prompt = `<s>[INST] ${userMessage} [/INST]`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);

// Learn more about the Llama 2 prompt format at:
```

```
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-2
```

- Para obter detalhes da API, consulte InvokeModela Referência AWS SDK for JavaScript da API.

InvokeModel: Lhama 3

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Meta Llama 3 usando a API Invoke Model.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto.

```
// Send a prompt to Meta Llama 3 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
```

```
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-
tokens-used-with-meta-llama-3
```

- Para obter detalhes da API, consulte InvokeModela Referência AWS SDK for JavaScript da API.

InvokeModelWithResponseTransmissão: Llama 2

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Meta Llama 2 usando a API Invoke Model e imprimir o fluxo de resposta.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto e processar o fluxo de resposta em tempo real.

```
// Send a prompt to Meta Llama 2 and print the response stream in real-time.

import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 2 Chat 13B.
const modelId = "meta.llama2-13b-chat-v1";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 2's prompt format.
const prompt = `<s>[INST] ${userMessage} [/INST]`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
```

```
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Extract and print the response stream in real-time.
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
  const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  if (chunk.generation) {
    process.stdout.write(chunk.generation);
  }
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-
tokens-used-with-meta-llama-3
```

- Para obter detalhes da API, consulte InvokeModelWithResponseStream na Referência AWS SDK for JavaScript da API.

InvokeModelWithResponseTransmissão: Llama 3

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Meta Llama 3, usando a API Invoke Model, e imprimir o fluxo de resposta.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto e processar o fluxo de resposta em tempo real.

```
// Send a prompt to Meta Llama 3 and print the response stream in real-time.

import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
```

```
    modelId,
  }),
);

// Extract and print the response stream in real-time.
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
  const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  if (chunk.generation) {
    process.stdout.write(chunk.generation);
  }
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- Para obter detalhes da API, consulte InvokeModelWithResponseStream na Referência AWS SDK for JavaScript da API.

IA Mistral

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Mistral usando a API Converse do Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para Mistral usando a API Converse do Bedrock.

```
// Use the Conversation API to send a text message to Mistral.

import {
```

```
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte Converse em Referência de AWS SDK for JavaScript API.

ConverseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto para a Mistral usando a API Converse da Bedrock e processar o fluxo de resposta em tempo real.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para Mistral usando a API Converse da Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
```

```
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK for JavaScript da API.

InvokeModel

O exemplo de código a seguir mostra como enviar uma mensagem de texto para modelos Mistral, usando a API Invoke Model.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use a API Invoke Model para enviar uma mensagem de texto.

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

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes a Mistral 7B Instruct model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "mistral.mistral-7b-instruct-v0:2".
 */
export const invokeModel = async (
  prompt,
  modelId = "mistral.mistral-7b-instruct-v0:2",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `<s>[INST] ${prompt} [/INST]`;

  // Prepare the payload.
  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
```

```
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.outputs[0].text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.MISTRAL_7B.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- Para obter detalhes da API, consulte InvokeModela Referência AWS SDK for JavaScript da API.

Cenários

Invocar vários modelos de base no Amazon Bedrock

O exemplo de código a seguir mostra como preparar e enviar uma solicitação para uma variedade de modelos de linguagem grande (LLMs) no Amazon Bedrock

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

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

import { fileURLToPath } from "url";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { FoundationModels } from "../config/foundation_models.js";

/**
 * @typedef {Object} ModelConfig
 * @property {Function} module
 * @property {Function} invoker
 * @property {string} modelId
 * @property {string} modelName
 */

const greeting = new ScenarioOutput(
  "greeting",
  "Welcome to the Amazon Bedrock Runtime client demo!",
  { header: true },
);

const selectModel = new ScenarioInput("model", "First, select a model:", {
  type: "select",
  choices: Object.values(FoundationModels).map((model) => ({
    name: model.modelName,
    value: model,
  })),
});
```

```
const enterPrompt = new ScenarioInput("prompt", "Now, enter your prompt:", {
  type: "input",
});

const printDetails = new ScenarioOutput(
  "print details",
  /**
   * @param {{ model: ModelConfig, prompt: string }} c
   */
  (c) => console.log(`Invoking ${c.model.modelName} with '${c.prompt}'...`),
  { slow: false },
);

const invokeModel = new ScenarioAction(
  "invoke model",
  /**
   * @param {{ model: ModelConfig, prompt: string, response: string }} c
   */
  async (c) => {
    const modelModule = await c.model.module();
    const invoker = c.model.invoker(modelModule);
    c.response = await invoker(c.prompt, c.model.modelId);
  },
);

const printResponse = new ScenarioOutput(
  "print response",
  /**
   * @param {{ response: string }} c
   */
  (c) => c.response,
  { slow: false },
);

const scenario = new Scenario("Amazon Bedrock Runtime Demo", [
  greeting,
  selectModel,
  enterPrompt,
  printDetails,
  invokeModel,
  printResponse,
]);

if (process.argv[1] === fileURLToPath(import.meta.url)) {
```

```
  scenario.run();
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - InvokeModel
  - InvokeModelWithResponseTransmitir

Exemplos de agentes do Amazon Bedrock usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com Agents for Amazon Bedrock.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, agentes do Amazon Bedrock

O exemplo de código a seguir mostra como começar a usar Agents for Amazon Bedrock.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  BedrockAgentClient,
  GetAgentCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * @typedef {Object} AgentSummary
 */

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 region.
 * It then retrieves a list of existing agents using the streamlined paginator
 approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
 * - Use of the Bedrock Agents client to initialize and communicate with the AWS
 service.
 * - Listing resources in a paginated response pattern.
 * - Accessing an individual resource using a command object.
 *
 * @returns {Promise<void>} A promise that resolves when the function has completed
 execution.
 */
export const main = async () => {
  const region = "us-east-1";

  console.log("=".repeat(68));

  console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
  const client = new BedrockAgentClient({ region });

  console.log(`Retrieving the list of existing agents...`);
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});
```

```
  /** @type {AgentSummary[]} */
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  console.log(`Found ${agentSummaries.length} agents in ${region}.`);

  if (agentSummaries.length > 0) {
    for (const agentSummary of agentSummaries) {
      const agentId = agentSummary.agentId;
      console.log("=".repeat(68));
      console.log(`Retrieving agent with ID: ${agentId}:`);
      console.log("-".repeat(68));

      const command = new GetAgentCommand({ agentId });
      const response = await client.send(command);
      const agent = response.agent;

      console.log(` Name: ${agent.agentName}`);
      console.log(` Status: ${agent.agentStatus}`);
      console.log(` ARN: ${agent.agentArn}`);
      console.log(` Foundation model: ${agent.foundationModel}`);
    }
  }
  console.log("=".repeat(68));
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - GetAgent
  - ListAgents

Tópicos

- Ações

Ações

`CreateAgent`

O código de exemplo a seguir mostra como usar `CreateAgent`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie um agente do .

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  CreateAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Creates an Amazon Bedrock Agent.
 *
 * @param {string} agentName - A name for the agent that you create.
 * @param {string} foundationModel - The foundation model to be used by the agent
 you create.
 * @param {string} agentResourceRoleArn - The ARN of the IAM role with permissions
 required by the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing details of the created agent.
 */
export const createAgent = async (
```

```
  agentName,
  foundationModel,
  agentResourceRoleArn,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  const command = new CreateAgentCommand({
    agentName,
    foundationModel,
    agentResourceRoleArn,
  });
  const response = await client.send(command);

  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentName and accountId, and roleName with a
 unique name for the new agent,
  // the id of your AWS account, and the name of an existing execution role that the
 agent can use inside your account.
  // For foundationModel, specify the desired model. Ensure to remove the brackets
 '[]' before adding your data.

  // A string (max 100 chars) that can include letters, numbers, dashes '-', and
 underscores '_'.
  const agentName = "[your-bedrock-agent-name]";

  // Your AWS account id.
  const accountId = "[123456789012]";

  // The name of the agent's execution role. It must be prefixed by
 `AmazonBedrockExecutionRoleForAgents_`.
  const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

  // The ARN for the agent's execution role.
  // Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
  const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

  // Specify the model for the agent. Change if a different model is preferred.
  const foundationModel = "anthropic.claude-v2";
```

```
  // Check for unresolved placeholders in agentName and roleArn.
  checkForPlaceholders([agentName, roleArn]);

  console.log(`Creating a new agent...`);

  const agent = await createAgent(agentName, foundationModel, roleArn);
  console.log(agent);
}
```

- Para obter detalhes da API, consulte CreateAgenta Referência AWS SDK for JavaScript da API.

`DeleteAgent`

O código de exemplo a seguir mostra como usar `DeleteAgent`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua um agente.

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
 *
```

```
 * @param {string} agentId - The unique identifier of the agent to delete.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-
agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
 and some additional metadata.
 */
export const deleteAgent = (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });
  const command = new DeleteAgentCommand({ agentId });
  return client.send(command);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets (`[]`) before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Deleting agent with ID ${agentId}...`);

  const response = await deleteAgent(agentId);
  console.log(response);
}
```

- Para obter detalhes da API, consulte DeleteAgenta Referência AWS SDK for JavaScript da API.

`GetAgent`

O código de exemplo a seguir mostra como usar `GetAgent`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha um agente.

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing the agent details.
 */
export const getAgent = async (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const command = new GetAgentCommand({ agentId });
  const response = await client.send(command);
  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
```

```
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Retrieving agent with ID ${agentId}...`);

  const agent = await getAgent(agentId);
  console.log(agent);
}
```

- Para obter detalhes da API, consulte GetAgenta Referência AWS SDK for JavaScript da API.

`ListAgentActionGroups`

O código de exemplo a seguir mostra como usar `ListAgentActionGroups`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os grupos de ação de um agente.

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  ListAgentActionGroupsCommand,
  paginateListAgentActionGroups,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
```

```
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  // Create a paginator configuration
  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const params = { agentId, agentVersion };

  const pages = paginateListAgentActionGroups(paginatorConfig, params);

  // Paginate until there are no more results
  const actionGroupSummaries = [];
  for await (const page of pages) {
    actionGroupSummaries.push(...page.actionGroupSummaries);
  }

  return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
```

```
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const actionGroupSummaries = [];
  do {
    const command = new ListAgentActionGroupsCommand({
      agentId,
      agentVersion,
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
    const response = await client.send(command);

    for (const actionGroup of response.actionGroupSummaries || []) {
      actionGroupSummaries.push(actionGroup);
    }

    nextToken = response.nextToken;
  } while (nextToken);

  return actionGroupSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId and agentVersion with an existing agent's
 id and version.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
```

```
  const agentId = "[ABC123DE45]";

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
 'DRAFT').
  const agentVersion = "[DRAFT]";

  // Check for unresolved placeholders in agentId and agentVersion.
  checkForPlaceholders([agentId, agentVersion]);

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using ListAgentActionGroupsCommand:",
  );

  for (const actionGroup of await listAgentActionGroupsWithCommandObject(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using the paginateListAgents function:",
  );
  for (const actionGroup of await listAgentActionGroupsWithPaginator(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }
}
```

- Para obter detalhes da API, consulte ListAgentActionGroupsa Referência AWS SDK for JavaScript da API.

`ListAgents`

O código de exemplo a seguir mostra como usar `ListAgents`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os agentes que pertencem a uma conta.

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

import { fileURLToPath } from "url";

import {
  BedrockAgentClient,
  ListAgentsCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
```

```
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 ListAgentsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const agentSummaries = [];
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
    const paginatedResponse = await client.send(command);

    agentSummaries.push(...(paginatedResponse.agentSummaries || []));

    nextToken = paginatedResponse.nextToken;
  } while (nextToken);

  return agentSummaries;
};

// Invoke main function if this file was run directly.
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
  console.log("Listing agents using ListAgentsCommand:");
  for (const agent of await listAgentsWithCommandObject()) {
    console.log(agent);
  }

  console.log("=".repeat(68));
  console.log("Listing agents using the paginateListAgents function:");
  for (const agent of await listAgentsWithPaginator()) {
    console.log(agent);
  }
}
```

- Para obter detalhes da API, consulte ListAgentsa Referência AWS SDK for JavaScript da API.

Exemplos de agentes para Amazon Bedrock Runtime usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com Agents for Amazon Bedrock Runtime.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`InvokeAgent`

O código de exemplo a seguir mostra como usar `InvokeAgent`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

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

import {
  BedrockAgentRuntimeClient,
  InvokeAgentCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
```

```
  // });

  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (let chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
      console.log(chunk);
      const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
      completion += decodedResponse;
    }

    return { sessionId: sessionId, completion };
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- Para obter detalhes da API, consulte InvokeAgenta Referência AWS SDK for JavaScript da API.

CloudWatch exemplos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com CloudWatch.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Ações

`DeleteAlarms`

O código de exemplo a seguir mostra como usar `DeleteAlarms`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
```

```
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteAlarmsa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
```

```
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteAlarmsa Referência AWS SDK for JavaScript da API.

`DescribeAlarmsForMetric`

O código de exemplo a seguir mostra como usar `DescribeAlarmsForMetric`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
```

```
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DescribeAlarmsForMetrica Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
```

```
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DescribeAlarmsForMetrica Referência AWS SDK for JavaScript da API.

`DisableAlarmActions`

O código de exemplo a seguir mostra como usar `DisableAlarmActions`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DisableAlarmAções na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.

- Para obter detalhes da API, consulte DisableAlarmAções na Referência AWS SDK for JavaScript da API.

`EnableAlarmActions`

O código de exemplo a seguir mostra como usar `EnableAlarmActions`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte EnableAlarmAções na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
```

```
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte EnableAlarmAções na Referência AWS SDK for JavaScript da API.

`ListMetrics`

O código de exemplo a seguir mostra como usar `ListMetrics`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";
```

```
export const main = () => {
  // Use the AWS console to see available namespaces and metric names. Custom
 metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
    Dimensions: [
      {
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

  return client.send(command);
};
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListMetricsa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListMetricsa Referência AWS SDK for JavaScript da API.

`PutMetricAlarm`

O código de exemplo a seguir mostra como usar `PutMetricAlarm`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";
```

```
const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
 the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.

- Para obter detalhes da API, consulte PutMetricAlarme na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutMetricAlarme na Referência AWS SDK for JavaScript da API.

`PutMetricData`

O código de exemplo a seguir mostra como usar `PutMetricData`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
```

```
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutMetricDados na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutMetricDados na Referência AWS SDK for JavaScript da API.

CloudWatch Exemplos de eventos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com CloudWatch Eventos.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Ações

`PutEvents`

O código de exemplo a seguir mostra como usar `PutEvents`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
        Source: "my.app",

        // The name of the event that is being sent.
```

```
        DetailType: "My Custom Event",

        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutEventsa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
```

```
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

cwevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutEventsa Referência AWS SDK for JavaScript da API.

`PutRule`

O código de exemplo a seguir mostra como usar `PutRule`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";
```

```
const run = async () => {
  // Request parameters for PutRule.
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/
API_PutRule.html#API_PutRule_RequestParameters
  const command = new PutRuleCommand({
    Name: process.env.CLOUDWATCH_EVENTS_RULE,

    // The event pattern for the rule.
    //  Example: {"source": ["my.app"]}
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,

    // The state of the rule. Valid values: ENABLED, DISABLED
    State: "ENABLED",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutRulea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

cwevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutRulea Referência AWS SDK for JavaScript da API.

`PutTargets`

O código de exemplo a seguir mostra como usar `PutTargets`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutTargetsCommand({
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,

    // The targets to add to the rule.
    Targets: [
      {
        Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
        // The ID of the target. Choose a unique ID for each target.
        Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Crie o cliente em um módulo separado e exporte-o.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutTargetsa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myCloudWatchEventsTarget",
    },
  ],
};

cwevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutTargetsa Referência AWS SDK for JavaScript da API.

CloudWatch Exemplos de registros usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o CloudWatch Logs.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Ações

`CreateLogGroup`

O código de exemplo a seguir mostra como usar `CreateLogGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
```

```
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Para obter detalhes da API, consulte CreateLogGrupo na Referência AWS SDK for JavaScript da API.

`DeleteLogGroup`

O código de exemplo a seguir mostra como usar `DeleteLogGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
```

```
  }
};

export default run();
```

- Para obter detalhes da API, consulte DeleteLogGrupo na Referência AWS SDK for JavaScript da API.

`DeleteSubscriptionFilter`

O código de exemplo a seguir mostra como usar `DeleteSubscriptionFilter`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Para obter detalhes da API, consulte DeleteSubscriptionFiltrar na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteSubscriptionFiltrar na Referência AWS SDK for JavaScript da API.

`DescribeLogGroups`

O código de exemplo a seguir mostra como usar `DescribeLogGroups`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups && page.logGroups.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }

  console.log(logGroups);
  return logGroups;
};
```

- Para obter detalhes da API, consulte DescribeLogGrupos na Referência AWS SDK for JavaScript da API.

`DescribeSubscriptionFilters`

O código de exemplo a seguir mostra como usar `DescribeSubscriptionFilters`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-
logs";
import { client } from "../libs/client.js";

const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Para obter detalhes da API, consulte DescribeSubscriptionFiltros na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cwl.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DescribeSubscriptionFiltros na Referência AWS SDK for JavaScript da API.

`GetQueryResults`

O código de exemplo a seguir mostra como usar `GetQueryResults`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
```

```
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }
```

- Para obter detalhes da API, consulte GetQueryResultados na referência AWS SDK for JavaScript da API.

`PutSubscriptionFilter`

O código de exemplo a seguir mostra como usar `PutSubscriptionFilter`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
    // An ARN of a same-account Kinesis stream, Kinesis Firehose
    // delivery stream, or Lambda function.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
SubscriptionFilters.html
    destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,

    // A name for the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

    // A filter pattern for subscribing to a filtered stream of log events.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
FilterAndPatternSyntax.html
    filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,
```

```
    // The name of the log group. Messages in this group matching the filter pattern
    // will be sent to the destination ARN.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Para obter detalhes da API, consulte PutSubscriptionFiltrar na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutSubscriptionFiltrar na Referência AWS SDK for JavaScript da API.

`StartLiveTail`

O código de exemplo a seguir mostra como usar `StartLiveTail`.

SDK para JavaScript (v3)

Inclua os arquivos necessários.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-
cloudwatch-logs";
```

Gerencie os eventos da sessão do Live Tail.

```
async function handleResponseAsync(response) {
    try {
      for await (const event of response.responseStream) {
        if (event.sessionStart !== undefined) {
          console.log(event.sessionStart);
        } else if (event.sessionUpdate !== undefined) {
          for (const logEvent of event.sessionUpdate.sessionResults) {
            const timestamp = logEvent.timestamp;
            const date = new Date(timestamp);
            console.log("[" + date + "] " + logEvent.message);
          }
        } else {
            console.error("Unknown event type");
        }
      }
    } catch (err) {
```

```
        // On-stream exceptions are captured here
        console.error(err)
    }
}
```

Inicie a sessão do Live Tail.

```
    const client = new CloudWatchLogsClient();

    const command = new StartLiveTailCommand({
        logGroupIdentifiers: logGroupIdentifiers,
        logStreamNames: logStreamNames,
        logEventFilterPattern: filterPattern
    });
    try{
        const response = await client.send(command);
        handleResponseAsync(response);
    } catch (err){
        // Pre-stream exceptions are captured here
        console.log(err);
    }
```

Interrompa a sessão do Live Tail após um período decorrido.

```
    /* Set a timeout to close the client. This will stop the Live Tail session. */
    setTimeout(function() {
        console.log("Client timeout");
        client.destroy();
      }, 10000);
```

- Para obter detalhes da API, consulte StartLiveTail in AWS SDK for JavaScript API Reference.

`StartQuery`

O código de exemplo a seguir mostra como usar `StartQuery`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
      // before the log group was created.
      throw new DateOutOfBoundsError(message);
    }

    throw err;
  }
}
```

- Para obter detalhes da API, consulte StartQuerya Referência AWS SDK for JavaScript da API.

Cenários

Executar uma consulta grande

O exemplo de código a seguir mostra como usar o CloudWatch Logs para consultar mais de 10.000 registros.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Esse é o ponto de entrada.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}

const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
  logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
  dateRange: [
    new Date(parseInt(process.env.QUERY_START_DATE)),
    new Date(parseInt(process.env.QUERY_END_DATE)),
  ],
});

await cloudWatchQuery.run();

console.log(
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found:
 ${cloudWatchQuery.results.length}`,
```

```
);
```

Essa é uma classe que divide as consultas em várias etapas, se necessário.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  StartQueryCommand,
  GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}

export class CloudWatchQuery {
  /**
   * Run a query for all CloudWatch Logs within a certain date range.
   * CloudWatch logs return a max of 10,000 results. This class
   * performs a binary search across all of the logs in the provided
   * date range if a query returns the maximum number of results.
   *
   * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient} client
   * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
 { limit: number } }} config
   */
  constructor(client, { logGroupNames, dateRange, queryConfig }) {
    this.client = client;
    /**
     * All log groups are queried.
     */
    this.logGroupNames = logGroupNames;

    /**
     * The inclusive date range that is queried.
     */
    this.dateRange = dateRange;

    /**
     * CloudWatch Logs never returns more than 10,000 logs.
     */
    this.limit = queryConfig?.limit ?? 10000;
```

```
  /**
   * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
   */
  this.results = [];
}

/**
 * Run the query.
 */
async run() {
  this.secondsElapsed = 0;
  const start = new Date();
  this.results = await this._largeQuery(this.dateRange);
  const end = new Date();
  this.secondsElapsed = (end - start) / 1000;
  return this.results;
}

/**
 * Recursively query for logs.
 * @param {[Date, Date]} dateRange
 * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[][]>}
 */
async _largeQuery(dateRange) {
  const logs = await this._query(dateRange, this.limit);

  console.log(
    `Query date range: ${dateRange
      .map((d) => d.toISOString())
      .join(" to ")}. Found ${logs.length} logs.`,
  );

  if (logs.length < this.limit) {
    return logs;
  }

  const lastLogDate = this._getLastLogDate(logs);
  const offsetLastLogDate = new Date(lastLogDate);
  offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
  const subDateRange = [offsetLastLogDate, dateRange[1]];
  const [r1, r2] = splitDateRange(subDateRange);
  const results = await Promise.all([
    this._largeQuery(r1),
```

```
    this._largeQuery(r2),
  ]);
  return [logs, ...results].flat();
}

/**
 * Find the most recent log in a list of logs.
 * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
 */
_getLastLogDate(logs) {
  const timestamps = logs
    .map(
      (log) =>
        log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
    )
    .filter((t) => !!t)
    .map((t) => `${t}Z`)
    .sort();

  if (!timestamps.length) {
    throw new Error("No timestamp found in logs.");
  }

  return new Date(timestamps[timestamps.length - 1]);
}

// snippet-start:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
// snippet-end:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]

/**
 * Starts a query and waits for it to complete.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 */
async _query(dateRange, maxLogs) {
  try {
    const { queryId } = await this._startQuery(dateRange, maxLogs);
```

```
    const { results } = await this._waitUntilQueryDone(queryId);
    return results ?? [];
  } catch (err) {
    /**
     * This error is thrown when StartQuery returns an error indicating
     * that the query's start or end date occur before the log group was
     * created.
     */
    if (err instanceof DateOutOfBoundsError) {
      return [];
    } else {
      throw err;
    }
  }
}

// snippet-start:[javascript.v3.cloudwatch-logs.actions.StartQuery]
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
      // before the log group was created.
      throw new DateOutOfBoundsError(message);
    }
```

```
      throw err;
    }
  }
  // snippet-end:[javascript.v3.cloudwatch-logs.actions.StartQuery]

  /**
   * Call GetQueryResultsCommand until the query is done.
   * @param {string} queryId
   */
  _waitUntilQueryDone(queryId) {
    const getResults = async () => {
      const results = await this._getQueryResults(queryId);
      const queryDone = [
        "Complete",
        "Failed",
        "Cancelled",
        "Timeout",
        "Unknown",
      ].includes(results.status);

      return { queryDone, results };
    };

    return retry(
      { intervalInMs: 1000, maxRetries: 60, quiet: true },
      async () => {
        const { queryDone, results } = await getResults();
        if (!queryDone) {
          throw new Error("Query not done.");
        }

        return results;
      },
    );
  }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - GetQueryResultados
  - StartQuery

CodeBuild exemplos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com CodeBuild.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Ações

`CreateProject`

O código de exemplo a seguir mostra como usar `CreateProject`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie um projeto.

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
```

```
} from "@aws-sdk/client-codebuild";

// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});

  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
        location: buildOutputBucket,
      },
      // Information about the build environment. The combination of "computeType"
 and "type" determines the
      // requirements for the environment such as CPU, memory, and disk space.
      environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
compute-types.html
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
      },
      name: projectName,
      // A role ARN with permission to create a CodeBuild project, write to the
 artifact location, and write CloudWatch logs.
      serviceRole: roleArn,
      source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be built.
        location: githubUrl,
      },
    }),
```

```
);
console.log(response);
//   {
//     '$metadata': {
//       httpStatusCode: 200,
//       requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
//       extendedRequestId: undefined,
//       cfId: undefined,
//       attempts: 1,
//       totalRetryDelay: 0
//     },
//     project: {
//       arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
//       artifacts: {
//         encryptionDisabled: false,
//         location: 'xxxxxx-xxxxxxx-xxxxxx',
//         name: 'MyCodeBuilder',
//         namespaceType: 'NONE',
//         packaging: 'NONE',
//         type: 'S3'
//       },
//       badge: { badgeEnabled: false },
//       cache: { type: 'NO_CACHE' },
//       created: 2023-08-18T14:46:48.979Z,
//       encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
//       environment: {
//         computeType: 'BUILD_GENERAL1_SMALL',
//         environmentVariables: [],
//         image: 'aws/codebuild/standard:7.0',
//         imagePullCredentialsType: 'CODEBUILD',
//         privilegedMode: false,
//         type: 'LINUX_CONTAINER'
//       },
//       lastModified: 2023-08-18T14:46:48.979Z,
//       name: 'MyCodeBuilder',
//       projectVisibility: 'PRIVATE',
//       queuedTimeoutInMinutes: 480,
//       serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
//       source: {
//         insecureSsl: false,
//         location: 'https://...',
//         reportBuildStatus: false,
//         type: 'GITHUB'
//       },
```

```
  //       timeoutInMinutes: 60
  //     }
  //   }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateProjecta Referência AWS SDK for JavaScript da API.

Exemplos de provedores de identidade Amazon Cognito usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Cognito Identity Provider.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Amazon Cognito

Os exemplos de código a seguir mostram como começar a usar o Amazon Cognito.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- Para obter detalhes da API, consulte ListUserPools na Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários

Ações

`AdminGetUser`

O código de exemplo a seguir mostra como usar `AdminGetUser`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
    Username: username,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte AdminGetUsuário na Referência AWS SDK for JavaScript da API.

`AdminInitiateAuth`

O código de exemplo a seguir mostra como usar `AdminInitiateAuth`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
```

```
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte AdminInitiateAuth na Referência AWS SDK for JavaScript da API.

`AdminRespondToAuthChallenge`

O código de exemplo a seguir mostra como usar `AdminRespondToAuthChallenge`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
  username,
  totp,
  session,
}) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AdminRespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: totp,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
```

```
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte AdminRespondToAuthDesafio na Referência AWS SDK for JavaScript da API.

`AssociateSoftwareToken`

O código de exemplo a seguir mostra como usar `AssociateSoftwareToken`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte AssociateSoftwareToken na Referência AWS SDK for JavaScript da API.

`ConfirmDevice`

O código de exemplo a seguir mostra como usar `ConfirmDevice`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmDeviceCommand({
    DeviceKey: deviceKey,
    AccessToken: accessToken,
    DeviceSecretVerifierConfig: {
      PasswordVerifier: passwordVerifier,
      Salt: salt,
    },
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte ConfirmDevicea Referência AWS SDK for JavaScript da API.

`ConfirmSignUp`

O código de exemplo a seguir mostra como usar `ConfirmSignUp`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const confirmSignUp = ({ clientId, username, code }) => {
```

```
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte ConfirmSignUp in AWS SDK for JavaScript API Reference.

`InitiateAuth`

O código de exemplo a seguir mostra como usar `InitiateAuth`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const initiateAuth = ({ username, password, clientId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
    AuthParameters: {
      USERNAME: username,
      PASSWORD: password,
    },
    ClientId: clientId,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte InitiateAutha Referência AWS SDK for JavaScript da API.

`ListUsers`

O código de exemplo a seguir mostra como usar `ListUsers`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte ListUsersa Referência AWS SDK for JavaScript da API.

`ResendConfirmationCode`

O código de exemplo a seguir mostra como usar `ResendConfirmationCode`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte ResendConfirmationCódigo na Referência AWS SDK for JavaScript da API.

`RespondToAuthChallenge`

O código de exemplo a seguir mostra como usar `RespondToAuthChallenge`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
```

```
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte RespondToAuthChallengea Referência AWS SDK for JavaScript da API.

`SignUp`

O código de exemplo a seguir mostra como usar `SignUp`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte SignUpa Referência AWS SDK for JavaScript da API.

`VerifySoftwareToken`

O código de exemplo a seguir mostra como usar `VerifySoftwareToken`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte VerifySoftwareToken na Referência AWS SDK for JavaScript da API.

Cenários

Inscrever um usuário em um grupo de usuários que exija MFA

O exemplo de código a seguir mostra como:

- Inscrever e confirmar um usuário com nome de usuário, senha e endereço de e-mail.
- Configurar a autenticação multifator associando uma aplicação de MFA ao usuário.
- Fazer login usando uma senha e um código de MFA.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Para obter a melhor experiência, clone o GitHub repositório e execute este exemplo. O código a seguir representa uma amostra da aplicação de exemplo completa.

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up'
 command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;
```

```
  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Signing up.`);
    await signUp({ clientId, username, password, email });
    log(`Signed up. A confirmation email has been sent to: ${email}.`);
    log(`Run 'confirm-sign-up ${username} <code>' to confirm your account.`);
  } catch (err) {
    log(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
```

```
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-
sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Confirming user.`);
    await confirmSignUp({ clientId, username, code });
    log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
    );
  } catch (err) {
    log(err);
  }
};

export { confirmSignUpHandler };
```

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../../actions/associate-software-
token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-
token' to finish the setup.",
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
```

```
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-initiate-
auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);

    log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }
```

```
    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    log(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../actions/admin-respond-to-auth-
challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-
respond-to-auth-challenge' command.`,
    );
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
```

```
      `Time-based one-time password (TOTP) is missing. It must be provided as an
 argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    log("Successfully authenticated.");
  } catch (err) {
    log(err);
  }
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
```

```
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-
software-token' command.`,
    );
  }
};
const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;

  try {
    validateTotp(totp);

    log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    console.log(err);
  }
};

export { verifySoftwareTokenHandler };
```

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - AdminGetUsuário
  - AdminInitiateAutenticação
  - AdminRespondToAuthDesafio
  - AssociateSoftwareSímbolo
  - ConfirmDevice
  - ConfirmSignPara cima
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCódigo
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareSímbolo

Exemplos do Amazon DocumentDB usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon DocumentDB.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Exemplos sem servidor

Invocar uma função Lambda a partir de um gatilho do Amazon DocumentDB

O exemplo de código a seguir mostra como implementar uma função Lambda que recebe um evento acionado pelo recebimento de registros de um stream de alterações do DocumentDB. A função recupera a carga do DocumentDB e registra o conteúdo do registro.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do Amazon DocumentDB com o uso do Lambda. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
```

```
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

Exemplos do DynamoDB usando o SDK JavaScript para (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o DynamoDB.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, DynamoDB

O exemplo de código a seguir mostra como começar a usar o DynamoDB.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Para obter detalhes da API, consulte ListTablesa Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários
- Exemplos sem servidor

Ações

`BatchExecuteStatement`

O código de exemplo a seguir mostra como usar `BatchExecuteStatement`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie um lote de itens usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
```

```
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Obtenha um lote de itens usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
```

```
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Atualize um lote de itens usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Exclua um lote de itens usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte BatchExecuteDeclaração na Referência AWS SDK for JavaScript da API.

`BatchGetItem`

O código de exemplo a seguir mostra como usar `BatchGetItem`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte BatchGet.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses["Books"]);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte BatchGetItem na referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
```

```
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte BatchGetItem na referência AWS SDK for JavaScript da API.

`BatchWriteItem`

O código de exemplo a seguir mostra como usar `BatchWriteItem`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte BatchWrite.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year' is
 recommended
        // to account for duplicate titles.
        ["BatchWriteMoviesTable"]: putRequests,
      },
    });

    await docClient.send(command);
  }
};
```

- Para obter detalhes da API, consulte BatchWriteItem na referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.

- Para obter detalhes da API, consulte BatchWriteItem na referência AWS SDK for JavaScript da API.

`CreateTable`

O código de exemplo a seguir mostra como usar `CreateTable`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
```

```
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateTablea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
```

```
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateTablea Referência AWS SDK for JavaScript da API.

`DeleteItem`

O código de exemplo a seguir mostra como usar `DeleteItem`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte DeleteCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteItema Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Exclua um item de uma tabela.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Exclua um item de uma tabela usando o cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};
```

```
docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteItema Referência AWS SDK for JavaScript da API.

`DeleteTable`

O código de exemplo a seguir mostra como usar `DeleteTable`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte DeleteTablea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteTablea Referência AWS SDK for JavaScript da API.

`DescribeTable`

O código de exemplo a seguir mostra como usar `DescribeTable`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DescribeTablea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DescribeTablea Referência AWS SDK for JavaScript da API.

`DescribeTimeToLive`

O código de exemplo a seguir mostra como usar `DescribeTimeToLive`.

SDK para JavaScript (v3)

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

const describeTableTTL = async (tableName, region) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    try {
        const ttlDescription = await client.send(new
 DescribeTimeToLiveCommand({ TableName: tableName }));
```

```
        if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED') {
            console.log("TTL is enabled for table %s.", tableName);
        } else {
            console.log("TTL is not enabled for table %s.", tableName);
        }

        return ttlDescription;
    } catch (e) {
        console.error(`Error describing table: ${e}`);
        throw e;
    }
}

// enter table name and change region if desired.
describeTableTTL('your-table-name', 'us-east-1');
```

- Para obter detalhes da API, consulte DescribeTimeToLivea Referência AWS SDK for JavaScript da API.

`ExecuteStatement`

O código de exemplo a seguir mostra como usar `ExecuteStatement`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie um item usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
```

```
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Obtenha um item usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Atualize um item usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Exclua um item usando o PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte ExecuteStatementa Referência AWS SDK for JavaScript da API.

`GetItem`

O código de exemplo a seguir mostra como usar `GetItem`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte GetItema Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Obtenha um item de uma tabela.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Obtenha um item de uma tabela usando o cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetItema Referência AWS SDK for JavaScript da API.

`ListTables`

O código de exemplo a seguir mostra como usar `ListTables`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
```

```
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListTablesa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListTablesa Referência AWS SDK for JavaScript da API.

`PutItem`

O código de exemplo a seguir mostra como usar `PutItem`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte PutItema Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Coloque um item em uma tabela.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Coloque um item em uma tabela usando o cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
```

```
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutItema Referência AWS SDK for JavaScript da API.

`Query`

O código de exemplo a seguir mostra como usar `Query`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
```

```
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for JavaScript .

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};
```

```
docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for JavaScript .

`Scan`

O código de exemplo a seguir mostra como usar `Scan`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte ScanCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
```

```
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK for JavaScript .

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte Scan na Referência da API AWS SDK for JavaScript .

`UpdateItem`

O código de exemplo a seguir mostra como usar `UpdateItem`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Neste exemplo, é usado o cliente de documentos para simplificar o trabalho com itens no DynamoDB. Para obter detalhes da API, consulte UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
```

```
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte UpdateItema Referência AWS SDK for JavaScript da API.

`UpdateTimeToLive`

O código de exemplo a seguir mostra como usar `UpdateTimeToLive`.

SDK para JavaScript (v3)

Ative o TTL em uma tabela existente do DynamoDB.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
```

```
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Desative o TTL em uma tabela existente do DynamoDB.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const disableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL disabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to disable TTL for table ${tableName}, response
 object: ${response}`);
```

```
        }
        return response;
    } catch (e) {
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Para obter detalhes da API, consulte UpdateTimeToLivea Referência AWS SDK for JavaScript da API.

Cenários

Atualizar condicionalmente o TTL de um item

O exemplo de código a seguir mostra como atualizar condicionalmente o TTL de um item.

SDK para JavaScript (v3)

Atualize o TTL em um item existente do DynamoDB em uma tabela, com uma condição.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

const updateDynamoDBItem = async (tableName, region, partitionKey, sortKey,
 newAttribute) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
```

```
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Enter your values here
updateDynamoDBItem('your-table-name', "us-east-1",'your-partition-key-value', 'your-
sort-key-value', 'your-new-attribute-value');
```

- Para obter detalhes da API, consulte UpdateItema Referência AWS SDK for JavaScript da API.

Crie um item com um TTL

O exemplo de código a seguir mostra como criar um item com TTL.

SDK para JavaScript (v3)

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
```

```
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 * 1000) /
 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, ex);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// use your own values
createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Para obter detalhes da API, consulte PutItema Referência AWS SDK for JavaScript da API.

Conceitos básicos de tabelas, itens e consultas

O exemplo de código a seguir mostra como:

- Criar uma tabela que possa conter dados de filmes.
- Colocar, obter e atualizar um único filme na tabela.
- Gravar dados de filmes na tabela usando um arquivo JSON de exemplo.
- Consultar filmes que foram lançados em determinado ano.
- Verificar filmes que foram lançados em um intervalo de anos.
- Excluir um filme da tabela e, depois, excluir a tabela.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
import { readFileSync } from "fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
```

```
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });

  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
 1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
```

```
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);
```

```
/**
 * Delete a movie from the table.
 */

log("Deleting a single movie from the table.");
const deleteCommand = new DeleteCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
});
await client.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
}
log("Movies added.");

/**
```

```
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log(`Scan for movies released between 1980 and 1990`);
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression. Scan
 will
```

```
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem

- Query
- Scan
- UpdateItem

Consultar uma tabela usando lotes de instruções PartiQL

O exemplo de código a seguir mostra como:

- Obter um lote de itens executando várias instruções SELECT.
- Adicionar um lote de itens executando várias instruções INSERT.
- Atualizar um lote de itens executando várias instruções UPDATE.
- Excluir um lote de itens executando várias instruções DELETE.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Execute instruções PartiQL em lote.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
```

```
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
```

```
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log(`Cities inserted.`);

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */
```

```
  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log(`Updated cities.`);

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */
```

```
  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obter detalhes da API, consulte BatchExecuteDeclaração na Referência AWS SDK for JavaScript da API.

Consultar uma tabela usando o PartiQL

O exemplo de código a seguir mostra como:

- Obter um item executando uma instrução SELECT.
- Adicionar um item executando uma instrução INSERT.
- Atualizar um item executando a instrução UPDATE.
- Excluir um item executando uma instrução DELETE.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Execute instruções PartiQL individuais.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
```

```
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
```

```
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */
```

```
  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log(`Coffee inserted.`);

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log(`Updated coffee`);

  /**
   * Delete the item.
   */
```

```
  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obter detalhes da API, consulte ExecuteStatementa Referência AWS SDK for JavaScript da API.

Consulta de itens TTL

O exemplo de código a seguir mostra como consultar itens TTL.

SDK para JavaScript (v3)

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

async function queryDynamoDBItems(tableName, region, primaryKey) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
```

```
    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

//enter your own values here
queryDynamoDBItems('your-table-name', 'your-partition-key-value');
```

- Para obter detalhes da API, consulte Query na Referência da API AWS SDK for JavaScript .

Atualizar o TTL de um item

O exemplo de código a seguir mostra como atualizar o TTL de um item.

SDK para JavaScript (v3)

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";
```

```
async function updateDynamoDBItem(tableName, region, partitionKey, sortKey) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

//enter your values here
updateDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Para obter detalhes da API, consulte UpdateItema Referência AWS SDK for JavaScript da API.

Exemplos sem servidor

Invocar uma função do Lambda em um gatilho do DynamoDB

O exemplo de código a seguir mostra como implementar uma função Lambda que recebe um evento acionado pelo recebimento de registros de um stream do DynamoDB. A função recupera a carga útil do DynamoDB e registra em log o conteúdo do registro.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do DynamoDB com o uso do Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consumindo um evento do DynamoDB com o uso do Lambda. 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)}`);
};
```

Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB

O exemplo de código a seguir mostra como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um stream do DynamoDB. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatando falhas de itens em lote do DynamoDB com o uso do Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
    }
  }
```

```
  return { batchItemFailures: [] };
};
```

Relatando falhas de itens em lote do DynamoDB com o uso do Lambda. TypeScript

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

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

    const batchItemsFailures: DynamoDBBatchItemFailure[] = []
    let curRecordSequenceNumber

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

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

    return batchItemsFailures
}
```

Exemplos do Amazon EC2 usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon EC2.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Amazon EC2

Os exemplos de código a seguir mostram como começar a usar o Amazon EC2.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeSecurityGroupsCommand } from "@aws-sdk/client-ec2";

import { client } from "./libs/client.js";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  try {
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({}),
    );

    const securityGroupList = SecurityGroups.slice(0, 9)
      .map((sg) => ` • ${sg.GroupId}: ${sg.GroupName}`)
      .join("\n");

    console.log(
      "Hello, Amazon EC2! Let's list up to 10 of your security groups:",
    );
    console.log(securityGroupList);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeSecurityGrupos na Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários

Ações

`AllocateAddress`

O código de exemplo a seguir mostra como usar `AllocateAddress`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { AllocateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new AllocateAddressCommand({});

  try {
    const { AllocationId, PublicIp } = await client.send(command);
    console.log("A new IP address has been allocated to your account:");
    console.log(`ID: ${AllocationId} Public IP: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2.
 Look under Network & Security > Elastic IPs",
    );
  } catch (err) {
    console.error(err);
  }
```

```
};
```

- Para obter detalhes da API, consulte AllocateAddressa Referência AWS SDK for JavaScript da API.

`AssociateAddress`

O código de exemplo a seguir mostra como usar `AssociateAddress`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { AssociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  // You need to allocate an Elastic IP address before associating it with an
 instance.
  // You can do that with the AllocateAddressCommand.
  const allocationId = "ALLOCATION_ID";
  // You need to create an EC2 instance before an IP address can be associated with
 it.
  // You can do that with the RunInstancesCommand.
  const instanceId = "INSTANCE_ID";
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  try {
    const { AssociationId } = await client.send(command);
    console.log(
      `Address with allocation ID ${allocationId} is now associated with instance
 ${instanceId}.`,
```

```
      `The association ID is ${AssociationId}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte AssociateAddressa Referência AWS SDK for JavaScript da API.

`AuthorizeSecurityGroupIngress`

O código de exemplo a seguir mostra como usar `AuthorizeSecurityGroupIngress`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { AuthorizeSecurityGroupIngressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Grant permissions for a single IP address to ssh into instances
// within the provided security group.
export const main = async () => {
  const command = new AuthorizeSecurityGroupIngressCommand({
    // Replace with a security group ID from the AWS console or
    // the DescribeSecurityGroupsCommand.
    GroupId: "SECURITY_GROUP_ID",
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        // Replace 0.0.0.0 with the IP address to authorize.
        // For more information on this notation, see
```

```
        // https://en.wikipedia.org/wiki/Classless_Inter-
Domain_Routing#CIDR_notation
        IpRanges: [{ CidrIp: "0.0.0.0/32" }],
      },
    ],
  });

  try {
    const { SecurityGroupRules } = await client.send(command);
    console.log(JSON.stringify(SecurityGroupRules, null, 2));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte AuthorizeSecurityGroupIngressa Referência AWS SDK for JavaScript da API.

`CreateKeyPair`

O código de exemplo a seguir mostra como usar `CreateKeyPair`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a key pair in Amazon EC2.
    const { KeyMaterial, KeyName } = await client.send(
      // A unique name for the key pair. Up to 255 ASCII characters.
      new CreateKeyPairCommand({ KeyName: "KEY_PAIR_NAME" }),
```

```
    );
    // This logs your private key. Be sure to save it.
    console.log(KeyName);
    console.log(KeyMaterial);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte CreateKeyPair na referência AWS SDK for JavaScript da API.

`CreateLaunchTemplate`

O código de exemplo a seguir mostra como usar `CreateLaunchTemplate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
```

```
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
```

- Para obter detalhes da API, consulte CreateLaunchModelo na Referência AWS SDK for JavaScript da API.

`CreateSecurityGroup`

O código de exemplo a seguir mostra como usar `CreateSecurityGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateSecurityGroupCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: "SECURITY_GROUP_NAME",
    // Up to 255 characters in length.
    Description: "DESCRIPTION",
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte CreateSecurityGrupo na Referência AWS SDK for JavaScript da API.

`DeleteKeyPair`

O código de exemplo a seguir mostra como usar `DeleteKeyPair`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DeleteKeyPairCommand({
    KeyName: "KEY_PAIR_NAME",
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DeleteKeyPair na referência AWS SDK for JavaScript da API.

`DeleteLaunchTemplate`

O código de exemplo a seguir mostra como usar `DeleteLaunchTemplate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- Para obter detalhes da API, consulte DeleteLaunchModelo na Referência AWS SDK for JavaScript da API.

`DeleteSecurityGroup`

O código de exemplo a seguir mostra como usar `DeleteSecurityGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteSecurityGroupCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DeleteSecurityGroupCommand({
    GroupId: "GROUP_ID",
  });

  try {
```

```
    await client.send(command);
    console.log("Security group deleted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DeleteSecurityGrupo na Referência AWS SDK for JavaScript da API.

`DescribeAddresses`

O código de exemplo a seguir mostra como usar `DescribeAddresses`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeAddressesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: ["ALLOCATION_ID"],
  });

  try {
    const { Addresses } = await client.send(command);
    const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
    console.log("Elastic IP addresses:");
    console.log(addressList.join("\n"));
  } catch (err) {
    console.error(err);
  }
```

```
};
```

- Para obter detalhes da API, consulte DescribeAddressesa Referência AWS SDK for JavaScript da API.

`DescribeIamInstanceProfileAssociations`

O código de exemplo a seguir mostra como usar `DescribeIamInstanceProfileAssociations`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
```

- Para obter detalhes da API, consulte DescribeIamInstanceProfileAssociações na Referência AWS SDK for JavaScript da API.

`DescribeImages`

O código de exemplo a seguir mostra como usar `DescribeImages`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { paginateDescribeImages } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first i386 image available for EC2 instances.
export const main = async () => {
  // The paginate function is a wrapper around the base command.
  const paginator = paginateDescribeImages(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the base command.
    { client, pageSize: 25 },
    {
      // There are almost 70,000 images available. Be specific with your filtering
      // to increase efficiency.
      // See https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
ec2/interfaces/describeimagescommandinput.html#filters
      Filters: [{ Name: "architecture", Values: ["x86_64"] }],
    },
  );

  try {
    const arm64Images = [];
    for await (const page of paginator) {
      if (page.Images.length) {
        arm64Images.push(...page.Images);
        // Once we have at least 1 result, we can stop.
        if (arm64Images.length >= 1) {
          break;
        }
      }
    }
    console.log(arm64Images);
  } catch (err) {
```

```
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeImagesa Referência AWS SDK for JavaScript da API.

`DescribeInstanceTypes`

O código de exemplo a seguir mostra como usar `DescribeInstanceTypes`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import {
  paginateDescribeInstanceTypes,
  DescribeInstanceTypesCommand,
} from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first arm64 EC2 instance type available.
export const main = async () => {
  // The paginate function is a wrapper around the underlying command.
  const paginator = paginateDescribeInstanceTypes(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the underlying command.
    { client, pageSize: 25 },
    {
      Filters: [
        { Name: "processor-info.supported-architecture", Values: ["x86_64"] },
        { Name: "free-tier-eligible", Values: ["true"] },
      ],
    }
```

```
  );

  try {
    const instanceTypes = [];

    for await (const page of paginator) {
      if (page.InstanceTypes.length) {
        instanceTypes.push(...page.InstanceTypes);

        // When we have at least 1 result, we can stop.
        if (instanceTypes.length >= 1) {
          break;
        }
      }
    }
    console.log(instanceTypes);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeInstanceTipos na referência AWS SDK for JavaScript da API.

`DescribeInstances`

O código de exemplo a seguir mostra como usar `DescribeInstances`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";
```

```
// List all of your EC2 instances running with x86_64 architecture that were
// launched this month.
export const main = async () => {
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;
  const command = new DescribeInstancesCommand({
    Filters: [
      { Name: "architecture", Values: ["x86_64"] },
      { Name: "instance-state-name", Values: ["running"] },
      {
        Name: "launch-time",
        Values: [launchTimePattern],
      },
    ],
  });

  try {
    const { Reservations } = await client.send(command);
    const instanceList = Reservations.reduce((prev, current) => {
      return prev.concat(current.Instances);
    }, []);

    console.log(instanceList);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeInstancesa Referência AWS SDK for JavaScript da API.

`DescribeKeyPairs`

O código de exemplo a seguir mostra como usar `DescribeKeyPairs`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeKeyPairsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeKeyPairsCommand({});

  try {
    const { KeyPairs } = await client.send(command);
    const keyPairList = KeyPairs.map(
      (kp) => ` • ${kp.KeyPairId}: ${kp.KeyName}`,
    ).join("\n");
    console.log("The following key pairs were found in your account:");
    console.log(keyPairList);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeKeyPares na Referência AWS SDK for JavaScript da API.

`DescribeRegions`

O código de exemplo a seguir mostra como usar `DescribeRegions`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeRegionsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions true even the regions that require opt-in will be returned.
    AllRegions: true,
    // You can omit the Filters property if you want to get all regions.
    Filters: [
      {
        Name: "region-name",
        // You can specify multiple values for a filter.
        // You can also use '*' as a wildcard. This will return all
        // of the regions that start with `us-east-`.
        Values: ["ap-southeast-4"],
      },
    ],
  });

  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeRegionsa Referência AWS SDK for JavaScript da API.

`DescribeSecurityGroups`

O código de exemplo a seguir mostra como usar `DescribeSecurityGroups`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeSecurityGroupsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Log the details of a specific security group.
export const main = async () => {
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: ["SECURITY_GROUP_ID"],
  });

  try {
    const { SecurityGroups } = await client.send(command);
    console.log(JSON.stringify(SecurityGroups, null, 2));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeSecurityGrupos na Referência AWS SDK for JavaScript da API.

`DescribeSubnets`

O código de exemplo a seguir mostra como usar `DescribeSubnets`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  }),
);
```

- Para obter detalhes da API, consulte DescribeSubnetsa Referência AWS SDK for JavaScript da API.

`DescribeVpcs`

O código de exemplo a seguir mostra como usar `DescribeVpcs`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
```

```
    );
```

- Para obter detalhes da API, consulte DescribeVpcsa Referência AWS SDK for JavaScript da API.

`DisassociateAddress`

O código de exemplo a seguir mostra como usar `DisassociateAddress`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DisassociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Disassociate an Elastic IP address from an instance.
export const main = async () => {
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AssociationId: "ASSOCIATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DisassociateAddressa Referência AWS SDK for JavaScript da API.

`MonitorInstances`

O código de exemplo a seguir mostra como usar `MonitorInstances`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { MonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Turn on detailed monitoring for the selected instance.
// By default, metrics are sent to Amazon CloudWatch every 5 minutes.
// For a cost you can enable detailed monitoring which sends metrics every minute.
export const main = async () => {
  const command = new MonitorInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instancesBeingMonitored = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instancesBeingMonitored.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte MonitorInstancesa Referência AWS SDK for JavaScript da API.

`RebootInstances`

O código de exemplo a seguir mostra como usar `RebootInstances`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { RebootInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new RebootInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte RebootInstancesa Referência AWS SDK for JavaScript da API.

`ReleaseAddress`

O código de exemplo a seguir mostra como usar `ReleaseAddress`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { ReleaseAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AllocationId: "ALLOCATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully released address.");
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte ReleaseAddressa Referência AWS SDK for JavaScript da API.

`ReplaceIamInstanceProfileAssociation`

O código de exemplo a seguir mostra como usar `ReplaceIamInstanceProfileAssociation`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );
```

- Para obter detalhes da API, consulte ReplaceIamInstanceProfileAssociação na Referência AWS SDK for JavaScript da API.

`RunInstances`

O código de exemplo a seguir mostra como usar `RunInstances`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { RunInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Create a new EC2 instance.
export const main = async () => {
```

```
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: "KEY_PAIR_NAME",
    // Your security group.
    SecurityGroupIds: ["SECURITY_GROUP_ID"],
    // An x86_64 compatible image.
    ImageId: "ami-0001a0d1a04bfcc30",
    // An x86_64 compatible free-tier instance type.
    InstanceType: "t1.micro",
    // Ensure only 1 instance launches.
    MinCount: 1,
    MaxCount: 1,
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte RunInstancesa Referência AWS SDK for JavaScript da API.

`StartInstances`

O código de exemplo a seguir mostra como usar `StartInstances`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { StartInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";
```

```
export const main = async () => {
  const command = new StartInstancesCommand({
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte StartInstancesa Referência AWS SDK for JavaScript da API.

`StopInstances`

O código de exemplo a seguir mostra como usar `StopInstances`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { StopInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new StopInstancesCommand({
```

```
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte StopInstancesa Referência AWS SDK for JavaScript da API.

`TerminateInstances`

O código de exemplo a seguir mostra como usar `TerminateInstances`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { TerminateInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new TerminateInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });
```

```
  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte TerminateInstancesa Referência AWS SDK for JavaScript da API.

`UnmonitorInstances`

O código de exemplo a seguir mostra como usar `UnmonitorInstances`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new UnmonitorInstancesCommand({
    InstanceIds: ["i-09a3dfe7ae00e853f"],
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instanceMonitoringsList = InstanceMonitorings.map(
      (im) =>
```

```
        ` • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instanceMonitoringsList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte UnmonitorInstancesa Referência AWS SDK for JavaScript da API.

Cenários

Criar e gerenciar um serviço resiliente

O exemplo de código a seguir mostra como criar um serviço web com balanceamento de carga que retorna recomendações de livros, filmes e músicas. O exemplo mostra como o serviço responde a falhas e como é possível reestruturá-lo para gerar mais resiliência em caso de falhas.

- Use um grupo do Amazon EC2 Auto Scaling para criar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) com base em um modelo de execução e para manter o número de instâncias em um intervalo especificado.
- Gerencie e distribua solicitações HTTP com o Elastic Load Balancing.
- Monitore a integridade das instâncias em um grupo do Auto Scaling e encaminhe solicitações somente para instâncias íntegras.
- Execute um servidor Web Python em cada instância do EC2 para lidar com solicitações HTTP. O servidor Web responde com recomendações e verificações de integridade.
- Simule um serviço de recomendação com uma tabela do Amazon DynamoDB.
- Controle a resposta do servidor web às solicitações e verificações de saúde atualizando AWS Systems Manager os parâmetros.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Execute o cenário interativo em um prompt de comando.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

Criar etapas para implantar todos os recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
```

```
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
```

```
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
```

```
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
```

```
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  );
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
```

```
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
```

```
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
new ScenarioAction("createLaunchTemplate", async () => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(
    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
```

```
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
```

```
        MaxSize: 3,
      }),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
   */
  (state) =>
    MESSAGES.createdAutoScalingGroup
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
      .replace(
        "${AVAILABILITY_ZONE_NAMES}",
        state.availabilityZoneNames.join(", "),
      ),
),
new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
  type: "confirm",
}),
new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
new ScenarioAction("getVpc", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
```

```
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
```

```
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
```

```
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
```

```
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
```

```
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
```

```
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

Criar etapas para executar a demonstração.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
```

```
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
```

```
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
```

```
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
```

```
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
```

```
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
```

```
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
 state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
```

```
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
```

```
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
```

```
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
```

```
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
```

```
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Criar etapas para destruir todos os recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
```

```
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
```

```
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
```

```
      }),
    );
  }
}),
new ScenarioOutput("deletePolicyResult", (state) => {
  if (state.deletePolicyError) {
    console.error(state.deletePolicyError);
    return MESSAGES.deletePolicyError.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  } else {
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }
}),
new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.removeRoleFromInstanceProfileError = e;
  }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
```

```
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
```

```
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
```

```
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyRoleFromProfileError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
  if (state.detachSsmOnlyRoleFromProfileError) {
    console.error(state.detachSsmOnlyRoleFromProfileError);
    return MESSAGES.detachSsmOnlyRoleFromProfileError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  } else {
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }
}),
new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyCustomRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
  if (state.detachSsmOnlyCustomRolePolicyError) {
    console.error(state.detachSsmOnlyCustomRolePolicyError);
    return MESSAGES.detachSsmOnlyCustomRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
```

```
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
```

```
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
```

```
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGrupos
  - CreateAutoScalingGroup
  - CreateInstancePerfil
  - CreateLaunchModelo
  - CreateListener
  - CreateLoadBalanceador
  - CreateTargetGrupo
  - DeleteAutoScalingGroup
  - DeleteInstancePerfil
  - DeleteLaunchModelo
  - DeleteLoadBalanceador
  - DeleteTargetGrupo
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZonas
  - DescribeIamInstanceProfileAssociações
  - DescribeInstances
  - DescribeLoadBalanceadores
  - DescribeSubnets
  - DescribeTargetGrupos
  - DescribeTargetHealth
  - DescribeVpcs

- RebootInstances
- ReplaceIamInstanceProfileAssociação
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Começar a usar instâncias

O exemplo de código a seguir mostra como:

- Criar um par de chaves e um grupo de segurança.
- Selecionar uma imagem de máquina da Amazon (AMI) e um tipo de instância compatível e, em seguida, criar uma instância.
- Interromper e reiniciar a instância.
- Associar um endereço IP elástico à sua instância.
- Conectar-se à sua instância via SSH e, em seguida, limpar os recursos.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Execute um cenário interativo em um prompt de comando.

```
import { mkdtempSync, writeFileSync, rmSync } from "fs";
import { tmpdir } from "os";
import { join } from "path";
import { get } from "http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
```

```
  DeleteSecurityGroupCommand,
  DescribeInstancesCommand,
  DescribeKeyPairsCommand,
  DescribeSecurityGroupsCommand,
  DisassociateAddressCommand,
  EC2Client,
  paginateDescribeImages,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";
import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

const ec2Client = new EC2Client();
const ssmClient = new SSMClient();

const prompter = new Prompter();
const confirmMessage = "Continue?";
const tmpDirectory = mkdtempSync(join(tmpdir(), "ec2-scenario-tmp"));

const createKeyPair = async (keyPairName) => {
  // Create a key pair in Amazon EC2.
  const { KeyMaterial, KeyPairId } = await ec2Client.send(
    // A unique name for the key pair. Up to 255 ASCII characters.
    new CreateKeyPairCommand({ KeyName: keyPairName }),
  );

  // Save the private key in a temporary location.
  writeFileSync(`${tmpDirectory}/${keyPairName}.pem`, KeyMaterial, {
    mode: 0o400,
  });

  return KeyPairId;
};
```

```
const describeKeyPair = async (keyPairName) => {
  const command = new DescribeKeyPairsCommand({
    KeyNames: [keyPairName],
  });
  const { KeyPairs } = await ec2Client.send(command);
  return KeyPairs[0];
};

const createSecurityGroup = async (securityGroupName) => {
  const command = new CreateSecurityGroupCommand({
    GroupName: securityGroupName,
    Description: "A security group for the Amazon EC2 example.",
  });
  const { GroupId } = await ec2Client.send(command);
  return GroupId;
};

const allocateIpAddress = async () => {
  const command = new AllocateAddressCommand({});
  const { PublicIp, AllocationId } = await ec2Client.send(command);
  return { PublicIp, AllocationId };
};

const getLocalIpAddress = () => {
  return new Promise((res, rej) => {
    get("http://checkip.amazonaws.com", (response) => {
      let data = "";
      response.on("data", (chunk) => (data += chunk));
      response.on("end", () => res(data.trim()));
    }).on("error", (err) => {
      rej(err);
    });
  });
};

const authorizeSecurityGroupIngress = async (securityGroupId) => {
  const ipAddress = await getLocalIpAddress();
  const command = new AuthorizeSecurityGroupIngressCommand({
    GroupId: securityGroupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
```

```
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  await ec2Client.send(command);
  return ipAddress;
};

const describeSecurityGroup = async (securityGroupName) => {
  const command = new DescribeSecurityGroupsCommand({
    GroupNames: [securityGroupName],
  });
  const { SecurityGroups } = await ec2Client.send(command);

  return SecurityGroups[0];
};

const getAmznLinux2AMIs = async () => {
  const AMIs = [];
  for await (const page of paginateGetParametersByPath(
    {
      client: ssmClient,
    },
    { Path: "/aws/service/ami-amazon-linux-latest" },
  )) {
    page.Parameters.forEach((param) => {
      if (param.Name.includes("amzn2")) {
        AMIs.push(param.Value);
      }
    });
  }

  const imageDetails = [];

  for await (const page of paginateDescribeImages(
    { client: ec2Client },
    { ImageIds: AMIs },
  )) {
    imageDetails.push(...(page.Images || []));
  }

  const choices = imageDetails.map((image, index) => ({
    name: `${image.ImageId} - ${image.Description}`,
```

```
    value: index,
  }));

  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an image.",
    choices,
  });

  return imageDetails[selectedIndex];
};

/**
 * @param {import('@aws-sdk/client-ec2').Image} imageDetails
 */
const getCompatibleInstanceTypes = async (imageDetails) => {
  const paginator = paginateDescribeInstanceTypes(
    { client: ec2Client, pageSize: 25 },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          Values: [imageDetails.Architecture],
        },
        { Name: "instance-type", Values: ["*.micro", "*.small"] },
      ],
    },
  );

  const instanceTypes = [];

  for await (const page of paginator) {
    if (page.InstanceTypes.length) {
      instanceTypes.push(...(page.InstanceTypes || []));
    }
  }

  const choices = instanceTypes.map((type, index) => ({
    name: `${type.InstanceType} - Memory:${type.MemoryInfo.SizeInMiB}`,
    value: index,
  }));
```

```
  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an instance type.",
    choices,
  });
  return instanceTypes[selectedIndex];
};

const runInstance = async ({
  keyPairName,
  securityGroupId,
  imageId,
  instanceType,
}) => {
  const command = new RunInstancesCommand({
    KeyName: keyPairName,
    SecurityGroupIds: [securityGroupId],
    ImageId: imageId,
    InstanceType: instanceType,
    MinCount: 1,
    MaxCount: 1,
  });

  const { Instances } = await ec2Client.send(command);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [Instances[0].InstanceId] },
  );
  return Instances[0].InstanceId;
};

const describeInstance = async (instanceId) => {
  const command = new DescribeInstancesCommand({
    InstanceIds: [instanceId],
  });

  const { Reservations } = await ec2Client.send(command);
  return Reservations[0].Instances[0];
};

const displaySSHConnectionInfo = ({ publicIp, keyPairName }) => {
  return `ssh -i ${tmpDirectory}/${keyPairName}.pem ec2-user@${publicIp}`;
```

```
};

const stopInstance = async (instanceId) => {
  const command = new StopInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(command);
  await waitUntilInstanceStopped(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
};

const startInstance = async (instanceId) => {
  const startCommand = new StartInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(startCommand);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
  return await describeInstance(instanceId);
};

const associateAddress = async ({ allocationId, instanceId }) => {
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  const { AssociationId } = await ec2Client.send(command);
  return AssociationId;
};

const disassociateAddress = async (associationId) => {
  const command = new DisassociateAddressCommand({
    AssociationId: associationId,
  });
  try {
    await ec2Client.send(command);
  } catch (err) {
    console.warn(
      `Failed to disassociated address with association id: ${associationId}`,
      err,
    );
  }
};
```

```
const releaseAddress = async (allocationId) => {
  const command = new ReleaseAddressCommand({
    AllocationId: allocationId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Address with allocation ID ${allocationId} released.\n`);
  } catch (err) {
    console.log(
      `Failed to release address with allocation id: ${allocationId}.`,
      err,
    );
  }
};

const restartInstance = async (instanceId) => {
  console.log("Stopping instance.");
  await stopInstance(instanceId);
  console.log("Instance stopped.");
  console.log("Starting instance.");
  const { PublicIpAddress } = await startInstance(instanceId);
  return PublicIpAddress;
};

const terminateInstance = async (instanceId) => {
  const command = new TerminateInstancesCommand({
    InstanceIds: [instanceId],
  });

  try {
    await ec2Client.send(command);
    await waitUntilInstanceTerminated(
      { client: ec2Client },
      { InstanceIds: [instanceId] },
    );
    console.log(`Instance with ID ${instanceId} terminated.\n`);
  } catch (err) {
    console.warn(`Failed to terminate instance ${instanceId}.`, err);
  }
};

const deleteSecurityGroup = async (securityGroupId) => {
```

```
  const command = new DeleteSecurityGroupCommand({
    GroupId: securityGroupId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Security group ${securityGroupId} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete security group ${securityGroupId}.`, err);
  }
};

const deleteKeyPair = async (keyPairName) => {
  const command = new DeleteKeyPairCommand({
    KeyName: keyPairName,
  });

  try {
    await ec2Client.send(command);
    console.log(`Key pair ${keyPairName} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete key pair ${keyPairName}.`, err);
  }
};

const deleteTemporaryDirectory = () => {
  try {
    rmSync(tmpDirectory, { recursive: true });
    console.log(`Temporary directory ${tmpDirectory} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete temporary directory ${tmpDirectory}.`, err);
  }
};

export const main = async () => {
  const keyPairName = "ec2-scenario-key-pair";
  const securityGroupName = "ec2-scenario-security-group";

  let securityGroupId, ipAllocationId, publicIp, instanceId, associationId;

  console.log(wrapText("Welcome to the Amazon EC2 basic usage scenario."));

  try {
    // Prerequisites
```

```
    console.log(
      "Before you launch an instance, you'll need a few things:",
      "\n - A Key Pair",
      "\n - A Security Group",
      "\n - An IP Address",
      "\n - An AMI",
      "\n - A compatible instance type",
      "\n\n I'll go ahead and take care of the first three, but I'll need your help
for the rest.",
    );

    await prompter.confirm({ message: confirmMessage });

    await createKeyPair(keyPairName);
    securityGroupId = await createSecurityGroup(securityGroupName);
    const { PublicIp, AllocationId } = await allocateIpAddress();
    ipAllocationId = AllocationId;
    publicIp = PublicIp;
    const ipAddress = await authorizeSecurityGroupIngress(securityGroupId);

    const { KeyName } = await describeKeyPair(keyPairName);
    const { GroupName } = await describeSecurityGroup(securityGroupName);
    console.log(`# created the key pair ${KeyName}.\n`);
    console.log(
      `# created the security group ${GroupName}`,
      `and allowed SSH access from ${ipAddress} (your IP).\n`,
    );
    console.log(`# allocated ${publicIp} to be used for your EC2 instance.\n`);

    await prompter.confirm({ message: confirmMessage });

    // Creating the instance
    console.log(wrapText("Create the instance."));
    console.log(
      "You get to choose which image you want. Select an amazon-linux-2 image from
the following:",
    );
    const imageDetails = await getAmznLinux2AMIs();
    const instanceTypeDetails = await getCompatibleInstanceTypes(imageDetails);
    console.log("Creating your instance. This can take a few seconds.");
    instanceId = await runInstance({
      keyPairName,
      securityGroupId,
      imageId: imageDetails.ImageId,
```

```
      instanceType: instanceTypeDetails.InstanceType,
    });
    const instanceDetails = await describeInstance(instanceId);
    console.log(`# instance ${instanceId}.\n`);
    console.log(instanceDetails);
    console.log(
      `\nYou should now be able to SSH into your instance from another terminal:`,
      `\n${displaySSHConnectionInfo({
        publicIp: instanceDetails.PublicIpAddress,
        keyPairName,
      })}`,
    );

    await prompter.confirm({ message: confirmMessage });

    // Understanding the IP address.
    console.log(wrapText("Understanding the IP address."));
    console.log(
      "When you stop and start an instance, the IP address will change. I'll restart
your",
      "instance for you. Notice how the IP address changes.",
    );
    const ipAddressAfterRestart = await restartInstance(instanceId);
    console.log(
      `\n Instance started. The IP address changed from
${instanceDetails.PublicIpAddress} to ${ipAddressAfterRestart}`,
      `\n${displaySSHConnectionInfo({
        publicIp: ipAddressAfterRestart,
        keyPairName,
      })}`,
    );
    await prompter.confirm({ message: confirmMessage });
    console.log(
      `If you want to the IP address to be static, you can associate an allocated`,
      `IP address to your instance. I allocated ${publicIp} for you earlier, and now
I'll associate it to your instance.`,
    );
    associationId = await associateAddress({
      allocationId: ipAllocationId,
      instanceId,
    });
    console.log(
      "Done. Now you should be able to SSH using the new IP.\n",
      `${displaySSHConnectionInfo({ publicIp, keyPairName })}`,
```

```
    );
    await prompter.confirm({ message: confirmMessage });
    console.log(
      "I'll restart the server again so you can see the IP address remains the
 same.",
    );
    const ipAddressAfterAssociated = await restartInstance(instanceId);
    console.log(
      `Done. Here's your SSH info. Notice the IP address hasn't changed.`,
      `\n${displaySSHConnectionInfo({
        publicIp: ipAddressAfterAssociated,
        keyPairName,
      })}`,
    );
    await prompter.confirm({ message: confirmMessage });
  } catch (err) {
    console.error(err);
  } finally {
    // Clean up.
    console.log(wrapText("Clean up."));
    console.log("Now I'll clean up all of the stuff I created.");
    await prompter.confirm({ message: confirmMessage });
    console.log("Cleaning up. Some of these steps can take a bit of time.");
    await disassociateAddress(associationId);
    await terminateInstance(instanceId);
    await releaseAddress(ipAllocationId);
    await deleteSecurityGroup(securityGroupId);
    deleteTemporaryDirectory();
    await deleteKeyPair(keyPairName);
    console.log(
      "Done cleaning up. Thanks for staying until the end!",
      "If you have any feedback please use the feedback button in the docs",
      "or create an issue on GitHub.",
    );
  }
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - AllocateAddress
  - AssociateAddress

- AuthorizeSecurityGroupIngress
- CreateKeyPar
- CreateSecurityGrupo
- DeleteKeyPar
- DeleteSecurityGrupo
- DescribeImages
- DescribeInstanceTipos
- DescribeInstances
- DescribeKeyPares
- DescribeSecurityGrupos
- DisassociateAddress
- ReleaseAddress
- RunInstances
- StartInstances
- StopInstances
- TerminateInstances
- UnmonitorInstances

Elastic Load Balancing — Exemplos da versão 2 usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Elastic Load Balancing - Versão 2.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Elastic Load Balancing

Os exemplos de código a seguir mostram como começar a usar o Elastic Load Balancing.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

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

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Para obter detalhes da API, consulte DescribeLoadBalancers em Referência de AWS SDK for JavaScript API.

Tópicos

Ações

`CreateListener`

O código de exemplo a seguir mostra como usar `CreateListener`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const client = new ElasticLoadBalancingV2Client({});
const { Listeners } = await client.send(
  new CreateListenerCommand({
    LoadBalancerArn: state.loadBalancerArn,
    Protocol: state.targetGroupProtocol,
    Port: state.targetGroupPort,
    DefaultActions: [
      { Type: "forward", TargetGroupArn: state.targetGroupArn },
    ],
  }),
);
```

- Para obter detalhes da API, consulte CreateListenera Referência AWS SDK for JavaScript da API.

`CreateLoadBalancer`

O código de exemplo a seguir mostra como usar `CreateLoadBalancer`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new CreateLoadBalancerCommand({
    Name: NAMES.loadBalancerName,
    Subnets: state.subnets,
  }),
);
state.loadBalancerDns = LoadBalancers[0].DNSName;
state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
await waitUntilLoadBalancerAvailable(
  { client },
  { Names: [NAMES.loadBalancerName] },
);
```

- Para obter detalhes da API, consulte CreateLoadBalancer na Referência AWS SDK for JavaScript da API.

`CreateTargetGroup`

O código de exemplo a seguir mostra como usar `CreateTargetGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
```

- Para obter detalhes da API, consulte CreateTargetGrupo na Referência AWS SDK for JavaScript da API.

`DeleteLoadBalancer`

O código de exemplo a seguir mostra como usar `DeleteLoadBalancer`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
  const client = new ElasticLoadBalancingV2Client({});
  const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
  await client.send(
    new DeleteLoadBalancerCommand({
      LoadBalancerArn: loadBalancer.LoadBalancerArn,
    }),
  );
  await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
    const lb = await findLoadBalancer(NAMES.loadBalancerName);
    if (lb) {
```

```
        throw new Error("Load balancer still exists.");
      }
    });
```

- Para obter detalhes da API, consulte DeleteLoadBalancer na Referência AWS SDK for JavaScript da API.

`DeleteTargetGroup`

O código de exemplo a seguir mostra como usar `DeleteTargetGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const client = new ElasticLoadBalancingV2Client({});
try {
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    client.send(
      new DeleteTargetGroupCommand({
        TargetGroupArn: TargetGroups[0].TargetGroupArn,
      }),
    ),
  );
} catch (e) {
  state.deleteLoadBalancerTargetGroupError = e;
}
```

- Para obter detalhes da API, consulte DeleteTargetGrupo na Referência AWS SDK for JavaScript da API.

`DescribeLoadBalancers`

O código de exemplo a seguir mostra como usar `DescribeLoadBalancers`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

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

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Para obter detalhes da API, consulte DescribeLoadBalancers em Referência de AWS SDK for JavaScript API.

`DescribeTargetGroups`

O código de exemplo a seguir mostra como usar `DescribeTargetGroups`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- Para obter detalhes da API, consulte DescribeTargetGrupos na Referência AWS SDK for JavaScript da API.

`DescribeTargetHealth`

O código de exemplo a seguir mostra como usar `DescribeTargetHealth`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- Para obter detalhes da API, consulte DescribeTargetHealth in AWS SDK for JavaScript API Reference.

Cenários

Criar e gerenciar um serviço resiliente

O exemplo de código a seguir mostra como criar um serviço web com balanceamento de carga que retorna recomendações de livros, filmes e músicas. O exemplo mostra como o serviço responde a falhas e como é possível reestruturá-lo para gerar mais resiliência em caso de falhas.

- Use um grupo do Amazon EC2 Auto Scaling para criar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) com base em um modelo de execução e para manter o número de instâncias em um intervalo especificado.
- Gerencie e distribua solicitações HTTP com o Elastic Load Balancing.
- Monitore a integridade das instâncias em um grupo do Auto Scaling e encaminhe solicitações somente para instâncias íntegras.
- Execute um servidor Web Python em cada instância do EC2 para lidar com solicitações HTTP. O servidor Web responde com recomendações e verificações de integridade.
- Simule um serviço de recomendação com uma tabela do Amazon DynamoDB.
- Controle a resposta do servidor web às solicitações e verificações de saúde atualizando AWS Systems Manager os parâmetros.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Execute o cenário interativo em um prompt de comando.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};
```

```
// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

Criar etapas para implantar todos os recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
```

```
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
```

```
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );
```

```
    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
```

```
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
```

```
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
```

```
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
```

```
    }),
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({
        AvailabilityZones: state.availabilityZoneNames,
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
          LaunchTemplateName: NAMES.launchTemplateName,
          Version: "$Default",
        },
        MinSize: 3,
        MaxSize: 3,
      }),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
```

```
    */
  (state) =>
    MESSAGES.createdAutoScalingGroup
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
      .replace(
        "${AVAILABILITY_ZONE_NAMES}",
        state.availabilityZoneNames.join(", "),
      ),
),
new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
  type: "confirm",
}),
new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
new ScenarioAction("getVpc", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
```

```
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
```

```
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
```

```
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
```

```
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
```

```
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
```

```
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

Criar etapas para executar a demonstração.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
```

```
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
```

```
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
```

```
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];
```

```
/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
```

```
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
```

```
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );
```

```
        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
```

```
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
```

```
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
```

```
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
```

```
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Criar etapas para destruir todos os recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
```

```
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
```

```
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
```

```
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
```

```
      NAMES.instancePolicyName,
    );
  } else {
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }
}),
new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.removeRoleFromInstanceProfileError = e;
  }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteRoleCommand({
        RoleName: NAMES.instanceRoleName,
      }),
    );
  } catch (e) {
    state.deleteInstanceRoleError = e;
```

```
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
```

```
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
```

```
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
```

```
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
```

```
  } catch (e) {
    state.detachSsmOnlyRoleFromProfileError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
  if (state.detachSsmOnlyRoleFromProfileError) {
    console.error(state.detachSsmOnlyRoleFromProfileError);
    return MESSAGES.detachSsmOnlyRoleFromProfileError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  } else {
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }
}),
new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyCustomRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
  if (state.detachSsmOnlyCustomRolePolicyError) {
    console.error(state.detachSsmOnlyCustomRolePolicyError);
    return MESSAGES.detachSsmOnlyCustomRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  } else {
    return MESSAGES.detachedSsmOnlyCustomRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  }
}),
new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
  try {
```

```
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
```

```
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
```

```
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGrupos
  - CreateAutoScalingGroup
  - CreateInstancePerfil
  - CreateLaunchModelo
  - CreateListener
  - CreateLoadBalanceador
  - CreateTargetGrupo
  - DeleteAutoScalingGroup
  - DeleteInstancePerfil
  - DeleteLaunchModelo
  - DeleteLoadBalanceador
  - DeleteTargetGrupo
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZonas
  - DescribeIamInstanceProfileAssociações
  - DescribeInstances
  - DescribeLoadBalanceadores
  - DescribeSubnets
  - DescribeTargetGrupos
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociação
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

EventBridge exemplos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com EventBridge.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Ações

`PutEvents`

O código de exemplo a seguir mostra como usar `PutEvents`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import {
  EventBridgeClient,
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";
```

```
export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutEventsCommand({
      Entries: [
        {
          Detail: JSON.stringify({ greeting: "Hello there." }),
          DetailType: detailType,
          Resources: resources,
          Source: source,
        },
      ],
    }),
  );

  console.log("PutEvents response:");
  console.log(response);
  // PutEvents response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- Para obter detalhes da API, consulte PutEventsa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

ebevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- Para obter detalhes da API, consulte PutEventsa Referência AWS SDK for JavaScript da API.

`PutRule`

O código de exemplo a seguir mostra como usar `PutRule`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";

export const putRule = async (
  ruleName = "some-rule",
  source = "some-source",
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutRuleCommand({
      Name: ruleName,
      EventPattern: JSON.stringify({ source: [source] }),
      State: "ENABLED",
      EventBusName: "default",
    }),
  );

  console.log("PutRule response:");
  console.log(response);
  // PutRule response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/
EventBridgeTestRule-1696280037720'
  // }
  return response;
```

```
};
```

- Para obter detalhes da API, consulte PutRulea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

ebevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- Para obter detalhes da API, consulte PutRulea Referência AWS SDK for JavaScript da API.

`PutTargets`

O código de exemplo a seguir mostra como usar `PutTargets`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
import {
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   FailedEntries: [],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- Para obter detalhes da API, consulte PutTargetsa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myEventBridgeTarget",
    },
  ],
};

ebevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
```

```
  }
});
```

- Para obter detalhes da API, consulte PutTargetsa Referência AWS SDK for JavaScript da API.

AWS Glue exemplos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com AWS Glue.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá AWS Glue

O exemplo de código a seguir mostra como começar a usar o AWS Glue.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});

export const main = async () => {
```

```
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- Para obter detalhes da API, consulte ListJobsa Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários

Ações

`CreateCrawler`

O código de exemplo a seguir mostra como usar `CreateCrawler`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
```

```
    },
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte CreateCrawlera Referência AWS SDK for JavaScript da API.

`CreateJob`

O código de exemplo a seguir mostra como usar `CreateJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte CreateJoba Referência AWS SDK for JavaScript da API.

`DeleteCrawler`

O código de exemplo a seguir mostra como usar `DeleteCrawler`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte DeleteCrawlera Referência AWS SDK for JavaScript da API.

`DeleteDatabase`

O código de exemplo a seguir mostra como usar `DeleteDatabase`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});
```

```
  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte DeleteDatabasea Referência AWS SDK for JavaScript da API.

`DeleteJob`

O código de exemplo a seguir mostra como usar `DeleteJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte DeleteJoba Referência AWS SDK for JavaScript da API.

`DeleteTable`

O código de exemplo a seguir mostra como usar `DeleteTable`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte DeleteTablea Referência AWS SDK for JavaScript da API.

`GetCrawler`

O código de exemplo a seguir mostra como usar `GetCrawler`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
```

```
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetCrawlera Referência AWS SDK for JavaScript da API.

`GetDatabase`

O código de exemplo a seguir mostra como usar `GetDatabase`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetDatabasea Referência AWS SDK for JavaScript da API.

`GetDatabases`

O código de exemplo a seguir mostra como usar `GetDatabases`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetDatabasesa Referência AWS SDK for JavaScript da API.

`GetJob`

O código de exemplo a seguir mostra como usar `GetJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });
```

```
  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetJoba Referência AWS SDK for JavaScript da API.

`GetJobRun`

O código de exemplo a seguir mostra como usar `GetJobRun`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetJobExecutar na Referência AWS SDK for JavaScript da API.

`GetJobRuns`

O código de exemplo a seguir mostra como usar `GetJobRuns`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetJobExecuções na Referência AWS SDK for JavaScript da API.

`GetTables`

O código de exemplo a seguir mostra como usar `GetTables`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });
```

```
  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetTablesa Referência AWS SDK for JavaScript da API.

`ListJobs`

O código de exemplo a seguir mostra como usar `ListJobs`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- Para obter detalhes da API, consulte ListJobsa Referência AWS SDK for JavaScript da API.

`StartCrawler`

O código de exemplo a seguir mostra como usar `StartCrawler`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte StartCrawlera Referência AWS SDK for JavaScript da API.

`StartJobRun`

O código de exemplo a seguir mostra como usar `StartJobRun`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte StartJobExecutar na Referência AWS SDK for JavaScript da API.

Cenários

Começar a executar crawlers e trabalhos

O exemplo de código a seguir mostra como:

- Criar um crawler que rastreie um bucket público do Amazon S3 e gere um banco de dados de metadados formatado em CSV.
- Liste informações sobre bancos de dados e tabelas em seu AWS Glue Data Catalog.
- Criar um trabalho para extrair dados em CSV do bucket do S3, transformá-los e carregar a saída formatada em JSON em outro bucket do S3.
- Listar informações sobre execuções de tarefas, visualizar dados transformados e limpar recursos.

Para obter mais informações, consulte Tutorial: Introdução ao AWS Glue Studio.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie e execute um crawler que examine um bucket público do Amazon Simple Storage Service (Amazon S3) e gere um banco de dados de metadados que descreva os dados no formato CSV que encontrar.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
```

```
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
  } catch {
    return false;
  }
};

/**
 * @param {{ createCrawler: import('../../../actions/create-
crawler.js').createCrawler}} actions
 */
const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
```

```
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH,
    );

    log("Crawler created successfully.", { type: "success" });
  }

  return { ...context };
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-
glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }

  log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
  await wait(waitTimeInSeconds);
  return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
  ({ startCrawler, getCrawler }) =>
  async (context) => {
    log("Starting crawler.");
    await startCrawler(process.env.CRAWLER_NAME);
    log("Crawler started.", { type: "success" });
```

```
    log("Waiting for crawler to finish running. This can take a while.");
    await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
    log("Crawler ready.", { type: "success" });

    return { ...context };
  };
```

Liste informações sobre bancos de dados e tabelas em seu AWS Glue Data Catalog.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};

const makeGetDatabaseStep =
  ({ getDatabase }) =>
  async (context) => {
    const {
      Database: { Name },
    } = await getDatabase(process.env.DATABASE_NAME);
    log(`Database: ${Name}`);
    return { ...context };
  };

/**
```

```
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-
glue').GetTablesCommandOutput}} config
 */
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
```

Crie e execute um trabalho que extraia dados em CSV do bucket do Amazon S3 de origem, transforme-os removendo e renomeando campos, e carregue a saída formatada em JSON em outro bucket do Amazon S3.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
```

```
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput> }  getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }

  switch (JobRun.JobRunState) {
    case "FAILED":
    case "TIMEOUT":
    case "STOPPED":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "RUNNING":
      break;
```

```
    case "SUCCEEDED":
      return;
    default:
      throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
  }

  log(
    `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
  );
  await wait(waitTimeInSeconds);
  return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
 view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
```

```
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);

    return { ...context };
  };
```

Liste informações sobre execuções de tarefas e visualize alguns dos dados transformados.

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput>}
 getJobRun
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunsCommandOutput}
 getJobRuns
 */
```

```
/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
 *
 * @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
 */
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (/** @type { Context } */ context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };
```

Exclua todos os recursos criados pela demonstração.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
```

```
  });

  return client.send(command);
};

const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};

const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};

const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};

/**
 *
 * @param {import('../../../actions/delete-job.js').deleteJob} deleteJobFn
 * @param {string[]} jobNames
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
```

```
   * @type {{ selectedJobNames: string[] }}
   */
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error)),
    );
    log("Jobs deleted.", { type: "success" });
  }
};

/**
 * @param {{
 *   listJobs: import('../../../actions/list-jobs.js').listJobs,
 *   deleteJob: import('../../../actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

/**
 * @param {import('../../../actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
```

```
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

/**
 * @param {{
 *  getTables: import('../../../actions/get-tables.js').getTables,
 *  deleteTable: import('../../../actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}}} context
   */
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null }),
    );

    if (TableList && TableList.length > 0) {
      /**
       * @type {{ tableNames: string[] }}
       */
      const { tableNames } = await context.prompter.prompt({
        name: "tableNames",
        type: "checkbox",
        message: "Let's clean up tables. Select tables to delete.",
        choices: TableList.map((t) => t.Name),
      });

      if (tableNames.length === 0) {
        log("No tables selected.");
      } else {
        log("Deleting tables.");
        await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
        log("Tables deleted.", { type: "success" });
      }
    }

    return { ...context };
  };
```

```
/**
 * @param {import('../../../actions/delete-database.js').deleteDatabase}
 deleteDatabase
 * @param {string[]} databaseNames
 */
const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('../../../actions/get-databases.js').getDatabases
 *   deleteDatabase: import('../../../actions/delete-database.js').deleteDatabase
 * }} config
 */
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}} context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbNames: string[] }} */
      const { dbNames } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbNames.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
      }
    }

    return { ...context };
  };
```

```
const cleanUpCrawlerStep = async (context) => {
  log(`Deleting crawler.`);

  try {
    await deleteCrawler(process.env.CRAWLER_NAME);
    log("Crawler deleted.", { type: "success" });
  } catch (err) {
    if (err.name === "EntityNotFoundException") {
      log(`Crawler is already deleted.`);
    } else {
      throw err;
    }
  }

  return { ...context };
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobCorra
  - GetJobCorre
  - GetTables
  - ListJobs
  - StartCrawler

- StartJobCorra

HealthImaging exemplos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com HealthImaging.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá HealthImaging

Os exemplos de código a seguir mostram como começar a usar HealthImaging.

SDK para JavaScript (v3)

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- Para obter detalhes da API, consulte ListDatastoresa Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Tópicos

- Ações
- Cenários

Ações

`CopyImageSet`

O código de exemplo a seguir mostra como usar `CopyImageSet`.

SDK para JavaScript (v3)

Função de utilitário para copiar um conjunto de imagens.

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
```

```
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //      },
  //       datastoreId: 'xxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING',
  //             latestVersionId: '1',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //       },
  //       sourceImageSetProperties: {
  //             createdAt: 2023-09-22T14:49:26.427Z,
```

```
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

Copiar um conjunto de imagens sem um destino.

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

Copiar um conjunto de imagens com um destino.

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "4",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

- Para obter detalhes da API, consulte CopyImageDefinir na referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`CreateDatastore`

O código de exemplo a seguir mostra como usar `CreateDatastore`.

SDK para JavaScript (v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //    datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- Para obter detalhes da API, consulte CreateDatastorea Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`DeleteDatastore`

O código de exemplo a seguir mostra como usar `DeleteDatastore`.

SDK para JavaScript (v3)

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //       },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- Para obter detalhes da API, consulte DeleteDatastorea Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`DeleteImageSet`

O código de exemplo a seguir mostra como usar `DeleteImageSet`.

SDK para JavaScript (v3)

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
```

```
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Para obter detalhes da API, consulte DeleteImageDefinir na referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`GetDICOMImportJob`

O código de exemplo a seguir mostra como usar `GetDICOMImportJob`.

SDK para JavaScript (v3)

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
```

```
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- Para obter detalhes da API, consulte GetDICOM ImportJob na Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`GetDatastore`

O código de exemplo a seguir mostra como usar `GetDatastore`.

SDK para JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
```

```
/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response["datastoreProperties"];
};
```

- Para obter detalhes da API, consulte GetDatastorea Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`GetImageFrame`

O código de exemplo a seguir mostra como usar `GetImageFrame`.

SDK para JavaScript (v3)

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     contentType: 'application/octet-stream',
  //     imageFrameBlob: <ref *1> IncomingMessage {}
```

```
  // }
  return response;
};
```

- Para obter detalhes da API, consulte GetImageQuadro na Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`GetImageSet`

O código de exemplo a seguir mostra como usar `GetImageSet`.

SDK para JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
```

```
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Para obter detalhes da API, consulte GetImageDefinir na referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`GetImageSetMetadata`

O código de exemplo a seguir mostra como usar `GetImageSetMetadata`.

SDK para JavaScript (v3)

Função de utilitário para obter metadados do conjunto de imagens.

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
```

```
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

Obter metadados do conjunto de imagens sem versão.

```
try {
  await getImageSetMetadata(
    "metadata.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012"
  );
} catch (err) {
  console.log("Error", err);
}
```

Obter metadados do conjunto de imagens com versão.

```
try {
  await getImageSetMetadata(
    "metadata2.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.log("Error", err);
}
```

- Para obter detalhes da API, consulte GetImageSetMetadataa Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`ListDICOMImportJobs`

O código de exemplo a seguir mostra como usar `ListDICOMImportJobs`.

SDK para JavaScript (v3)

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
```

```
  // }
  // ]}

  return jobSummaries;
};
```

- Para obter detalhes da API, consulte ListDicom ImportJobs na Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`ListDatastores`

O código de exemplo a seguir mostra como usar `ListDatastores`.

SDK para JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
```

```
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- Para obter detalhes da API, consulte ListDatastoresa Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`ListImageSetVersions`

O código de exemplo a seguir mostra como usar `ListImageSetVersions`.

SDK para JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
```

```
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Para obter detalhes da API, consulte ListImageSetVersionsa Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`ListTagsForResource`

O código de exemplo a seguir mostra como usar `ListTagsForResource`.

SDK para JavaScript (v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
```

```
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- Para obter detalhes da API, consulte ListTagsForResourcea Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`SearchImageSets`

O código de exemplo a seguir mostra como usar `SearchImageSets`.

SDK para JavaScript (v3)

A função de utilitário para pesquisar conjuntos de imagens.

```
import {paginateSearchImageSets} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters - The
 search criteria filters.
 * @param { import('@aws-sdk/client-medical-imaging').Sort } sort - The search
 criteria sort.
 */
export const searchImageSets = async (
    datastoreId = "xxxxxxxx",
    searchCriteria = {}
) => {
```

```
    const paginatorConfig = {
        client: medicalImagingClient,
        pageSize: 50,
    };

    const commandParams = {
        datastoreId: datastoreId,
        searchCriteria: searchCriteria,
    };

    const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

    const imageSetsMetadataSummaries = [];
    for await (const page of paginator) {
        // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
        imageSetsMetadataSummaries.push(...page["imageSetsMetadataSummaries"]);
        console.log(page);
    }
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     },
    //     imageSetsMetadataSummaries: [
    //         {
    //             DICOMTags: [Object],
    //             createdAt: "2023-09-19T16:59:40.551Z",
    //             imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',
    //             updatedAt: "2023-09-19T16:59:40.551Z",
    //             version: 1
    //         }]
    // }

    return imageSetsMetadataSummaries;
};
```

Caso de uso nº 1: operador EQUAL.

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [{DICOMPatientId: "1234567"}],
                operator: "EQUAL",
            },
        ]
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

Caso de uso #2: BETWEEN operador usando DICOM StudyDate e StudyTime DICOM.

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {
                        DICOMStudyDateAndTime: {
                            DICOMStudyDate: "19900101",
                            DICOMStudyTime: "000000",
                        },
                    },
                    {
                        DICOMStudyDateAndTime: {
                            DICOMStudyDate: "20230901",
                            DICOMStudyTime: "000000",
                        },
                    },
                ],
                operator: "BETWEEN",
            },
        ]
```

```
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

Caso de uso nº 3: operador BETWEEN usando o createdAt. Os estudos de tempo foram previamente persistidos.

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {createdAt: new Date("1985-04-12T23:20:50.52Z")},
                    {createdAt: new Date()},
                ],
                operator: "BETWEEN",
            },
        ]
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

Caso de uso #4: operador EQUAL em DICOM SeriesInstance UID e BETWEEN em updatedAt e classifique a resposta em ordem ASC no campo updatedAt.

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {updatedAt: new Date("1985-04-12T23:20:50.52Z")},
```

```
                        {updatedAt: new Date()},
                    ],
                    operator: "BETWEEN",
                },
                {
                    values: [
                        {DICOMSeriesInstanceUID:
"1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
                    ],
                    operator: "EQUAL",
                },
            ],
            sort: {
                sortOrder: "ASC",
                sortField: "updatedAt",
            }
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

- Para obter detalhes da API, consulte SearchImageConjuntos na referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`StartDICOMImportJob`

O código de exemplo a seguir mostra como usar `StartDICOMImportJob`.

SDK para JavaScript (v3)

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
```

```
/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
 stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/"
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- Para obter detalhes da API, consulte StartDICOM ImportJob na AWS SDK for JavaScript Referência da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`TagResource`

O código de exemplo a seguir mostra como usar `TagResource`.

SDK para JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                     - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
```

```
  // }

  return response;
};
```

- Para obter detalhes da API, consulte TagResourcea Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`UntagResource`

O código de exemplo a seguir mostra como usar `UntagResource`.

SDK para JavaScript (v3)

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
```

```
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     }
  // }

  return response;
};
```

- Para obter detalhes da API, consulte UntagResourcea Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

`UpdateImageSetMetadata`

O código de exemplo a seguir mostra como usar `UpdateImageSetMetadata`.

SDK para JavaScript (v3)

```
import {UpdateImageSetMetadataCommand} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}') => {
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
```

```
            imageSetId: imageSetId,
            latestVersionId: latestVersionId,
            updateImageSetMetadataUpdates: updateMetadata
        })
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
};
```

Caso de uso #1: insira ou atualize um atributo.

```
        const insertAttributes =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "updatableAttributes":
                    new TextEncoder().encode(insertAttributes)
```

```
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

Caso de uso #2: Remova um atributo.

```
        // Attribute key and value must match the existing attribute.
        const remove_attribute =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "removableAttributes":
                    new TextEncoder().encode(remove_attribute)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

Caso de uso #3: remova uma instância.

```
        const remove_instance =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "Series": {
                        "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                            "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                            }
```

```
                    }
                }
            }
        });

    const updateMetadata = {
        "DICOMUpdates": {
            "removableAttributes":
                new TextEncoder().encode(remove_instance)
        }
    };

    await updateImageSetMetadata(datastoreID, imageSetID,
        versionID, updateMetadata);
```

- Para obter detalhes da API, consulte UpdateImageSetMetadataa Referência AWS SDK for JavaScript da API.

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Cenários

Começar a usar conjuntos de imagens e quadros de imagem

O exemplo de código a seguir mostra como importar arquivos DICOM e baixar molduras de imagem em HealthImaging.

A implementação é estruturada como um aplicativo de linha de comando de fluxo de trabalho.

- Configurar recursos para uma importação DICOM.
- Importe arquivos DICOM para um armazenamento de dados.
- Recupere os IDs do conjunto de imagens para o trabalho de importação.
- Recupere os IDs do quadro de imagem para o conjunto de imagens.
- Baixe, decodifique e verifique os quadros de imagem.
- Limpar recursos.

SDK para JavaScript (v3)

index.js- Orquestre etapas.

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

import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
```

```
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
```

```
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

deploy-steps.js- Implante recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});
```

```
const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
```

```
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName == state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
```

```
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

dataset-steps.js- Copie arquivos DICOM.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
```

```
const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
```

```
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);
```

```
export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js- Inicie a importação para o armazenamento de dados.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
```

```
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
```

```
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

image-set-steps.js- Obtenha IDs de conjuntos de imagens.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });
```

```
    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js- Obtenha IDs de quadros de imagem.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

import {
  ScenarioAction,
```

```
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */
```

```
/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);
```

```
export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
  { slow: false },
);
```

verify-steps.js- Verifique os quadros de imagem. A biblioteca AWS HealthImaging Pixel Data Verification foi usada para verificação.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
```

```
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
```

```
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (/** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreID;
      const imageSetId = metadata.ImageSetID;

      for (const [seriesInstanceUid, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceUid} and
 sop ${sopInstanceUid}`,
          );
          const child = spawn(
            "node",
            [
```

```
              verificationTool,
              datastoreId,
              imageSetId,
              seriesInstanceUid,
              sopInstanceUid,
            ],
            { stdio: "inherit" },
          );

          await new Promise((resolve, reject) => {
            child.on("exit", (code) => {
              if (code === 0) {
                resolve();
              } else {
                reject(
                  new Error(
                    `Verification tool exited with code ${code} for image set
 ${imageSetId}`,
                  ),
                );
              }
            });
          });
        }
      }
    }
  },
);
```

clean-up-steps.js- Destrua recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";
```

```
import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */
```

```
/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
```

```
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - DeleteImageConjunto
  - Obtenha o DICOM ImportJob
  - GetImageQuadro
  - GetImageSetMetadata
  - SearchImageConjuntos
  - Inicie o DICOM ImportJob

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Marcar um datastore

O exemplo de código a seguir mostra como marcar um armazenamento HealthImaging de dados.

SDK para JavaScript (v3)

Marcar um datastore.

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(datastoreArn, tags);
  } catch (e) {
    console.log(e);
  }
```

A função de utilitário para marcar um recurso.

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                    - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
```

```
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

Listar tags para um datastore.

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(datastoreArn);
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

A função de utilitário para listar as tags de um recurso.

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
```

```
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

Desmarcar um datastore.

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(datastoreArn, keys);
  } catch (e) {
    console.log(e);
  }
```

A função de utilitário para desmarcar um recurso.

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
```

```
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Marcar um conjunto de imagens

O exemplo de código a seguir mostra como marcar um conjunto de HealthImaging imagens.

SDK para JavaScript (v3)

Marcar um conjunto de imagens

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(imagesetArn, tags);
  } catch (e) {
    console.log(e);
  }
```

A função de utilitário para marcar um recurso.

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                     - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
```

```
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

Listar tags para um conjunto de imagens

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(imagesetArn);
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

A função de utilitário para listar as tags de um recurso.

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
```

```
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

Desmarcar um conjunto de imagens

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(imagesetArn, keys);
  } catch (e) {
    console.log(e);
  }
```

A função de utilitário para desmarcar um recurso.

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
```

```
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exemplos de IAM usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o IAM.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, IAM

O exemplo de código a seguir mostra como começar a usar o IAM.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
 async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
  for await (const page of paginator) {
    if (page.Policies) {
      page.Policies.forEach((p) => {
        console.log(`${p.PolicyName}`);
        policyCount++;
```

```
      });
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- Para obter detalhes da API, consulte ListPoliciesa Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários

Ações

`AttachRolePolicy`

O código de exemplo a seguir mostra como usar `AttachRolePolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Anexe a política.

```
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);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte AttachRolePolítica na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        console.log(
          "AmazonDynamoDBFullAccess is already attached to this role."
        );
```

```
        process.exit();
      }
    });
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.attachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte AttachRolePolítica na Referência AWS SDK for JavaScript da API.

`CreateAccessKey`

O código de exemplo a seguir mostra como usar `CreateAccessKey`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie a chave de acesso.

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateAccessChave na referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateAccessChave na referência AWS SDK for JavaScript da API.

`CreateAccountAlias`

O código de exemplo a seguir mostra como usar `CreateAccountAlias`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Criar o alias da conta.

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias - A unique name for the account alias.
 * @returns
 */
export const createAccountAlias = (alias) => {
  const command = new CreateAccountAliasCommand({
    AccountAlias: alias,
  });

  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateAccountAlias na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateAccountAlias na Referência AWS SDK for JavaScript da API.

`CreateGroup`

O código de exemplo a seguir mostra como usar `CreateGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte CreateGroupa Referência AWS SDK for JavaScript da API.

`CreateInstanceProfile`

O código de exemplo a seguir mostra como usar `CreateInstanceProfile`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
```

- Para obter detalhes da API, consulte CreateInstancePerfil na Referência AWS SDK for JavaScript da API.

`CreatePolicy`

O código de exemplo a seguir mostra como usar `CreatePolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie a política .

```
import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyName
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: "*",
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  });
```

```
  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreatePolicya Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
      ],
```

```
      Resource: "RESOURCE_ARN",
    },
  ],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

iam.createPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreatePolicya Referência AWS SDK for JavaScript da API.

`CreateRole`

O código de exemplo a seguir mostra como usar `CreateRole`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie a função.

```
import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
  const command = new CreateRoleCommand({
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {
            Service: "lambda.amazonaws.com",
          },
          Action: "sts:AssumeRole",
        },
      ],
    }),
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte CreateRolea Referência AWS SDK for JavaScript da API.

`CreateSAMLProvider`

O código de exemplo a seguir mostra como usar `CreateSAMLProvider`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
```

```
import * as path from "path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";

const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.
 * For more information on generating this document,
 see https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_providers_create_saml.html#samlstep1.
 */
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte CreateSAMLProvider na Referência da API AWS SDK for JavaScript .

`CreateServiceLinkedRole`

O código de exemplo a seguir mostra como usar `CreateServiceLinkedRole`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Criar uma função vinculada ao serviço.

```
import {
  CreateServiceLinkedRoleCommand,
  GetRoleCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-
that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/
latest/gr/aws-service-information.html.
    AWSServiceName: serviceName,
  });
  try {
    const response = await client.send(command);
    console.log(response);
    return response;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInputException" &&
      caught.message.includes(
        "Service role name AWSServiceRoleForElasticBeanstalk has been taken in this
 account",
```

```
      )
    ) {
      console.warn(caught.message);
      return client.send(
        new GetRoleCommand({ RoleName: "AWSServiceRoleForElasticBeanstalk" }),
      );
    } else {
      throw caught;
    }
  }
};
```

- Para obter detalhes da API, consulte CreateServiceLinkedRolea Referência AWS SDK for JavaScript da API.

`CreateUser`

O código de exemplo a seguir mostra como usar `CreateUser`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie o usuário.

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const createUser = (name) => {
  const command = new CreateUserCommand({ UserName: name });
  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateUsera Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateUsera Referência AWS SDK for JavaScript da API.

`DeleteAccessKey`

O código de exemplo a seguir mostra como usar `DeleteAccessKey`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua a chave de acesso.

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const deleteAccessKey = (userName, accessKeyId) => {
  const command = new DeleteAccessKeyCommand({
    AccessKeyId: accessKeyId,
    UserName: userName,
  });

  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteAccessChave na referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

iam.deleteAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteAccessChave na referência AWS SDK for JavaScript da API.

`DeleteAccountAlias`

O código de exemplo a seguir mostra como usar `DeleteAccountAlias`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua o alias da conta.

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias
 */
export const deleteAccountAlias = (alias) => {
  const command = new DeleteAccountAliasCommand({ AccountAlias: alias });

  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteAccountAlias na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteAccountAlias na Referência AWS SDK for JavaScript da API.

`DeleteGroup`

O código de exemplo a seguir mostra como usar `DeleteGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
    GroupName: groupName,
```

```
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte DeleteGroupa Referência AWS SDK for JavaScript da API.

`DeleteInstanceProfile`

O código de exemplo a seguir mostra como usar `DeleteInstanceProfile`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
```

- Para obter detalhes da API, consulte DeleteInstancePerfil na Referência AWS SDK for JavaScript da API.

`DeletePolicy`

O código de exemplo a seguir mostra como usar `DeletePolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua a política.

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- Para obter detalhes da API, consulte DeletePolicya Referência AWS SDK for JavaScript da API.

`DeleteRole`

O código de exemplo a seguir mostra como usar `DeleteRole`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua a função.

```
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);
};
```

- Para obter detalhes da API, consulte DeleteRolea Referência AWS SDK for JavaScript da API.

`DeleteRolePolicy`

O código de exemplo a seguir mostra como usar `DeleteRolePolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
```

```
  });
  return client.send(command);
};
```

- Para obter detalhes da API, consulte DeleteRolePolítica na Referência AWS SDK for JavaScript da API.

`DeleteSAMLProvider`

O código de exemplo a seguir mostra como usar `DeleteSAMLProvider`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
  const command = new DeleteSAMLProviderCommand({
    SAMLProviderArn: providerArn,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte DeleteSAMLProvider na Referência da API AWS SDK for JavaScript .

`DeleteServerCertificate`

O código de exemplo a seguir mostra como usar `DeleteServerCertificate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Excluir um certificado de servidor.

```
import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
  const command = new DeleteServerCertificateCommand({
    ServerCertificateName: certName,
  });

  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteServerCertificado na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteServerCertificado na Referência AWS SDK for JavaScript da API.

`DeleteServiceLinkedRole`

O código de exemplo a seguir mostra como usar `DeleteServiceLinkedRole`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Para obter detalhes da API, consulte DeleteServiceLinkedRolea Referência AWS SDK for JavaScript da API.

`DeleteUser`

O código de exemplo a seguir mostra como usar `DeleteUser`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua o usuário.

```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
  const command = new DeleteUserCommand({ UserName: name });
  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteUsera Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
```

```
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteUsera Referência AWS SDK for JavaScript da API.

`DetachRolePolicy`

O código de exemplo a seguir mostra como usar `DetachRolePolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Desanexe a política.

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const detachRolePolicy = (policyArn, roleName) => {
  const command = new DetachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });
```

```
  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DetachRolePolítica na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        var params = {
          PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
          RoleName: process.argv[2],
        };
        iam.detachRolePolicy(params, function (err, data) {
          if (err) {
            console.log("Unable to detach policy from role", err);
```

```
          } else {
            console.log("Policy detached from role successfully");
            process.exit();
          }
        });
      }
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DetachRolePolítica na Referência AWS SDK for JavaScript da API.

`GetAccessKeyLastUsed`

O código de exemplo a seguir mostra como usar `GetAccessKeyLastUsed`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha a chave de acesso.

```
import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} accessKeyId
 */
export const getAccessKeyLastUsed = async (accessKeyId) => {
  const command = new GetAccessKeyLastUsedCommand({
    AccessKeyId: accessKeyId,
  });
```

```
  const response = await client.send(command);

  if (response.AccessKeyLastUsed?.LastUsedDate) {
    console.log(`
    ${accessKeyId} was last used by ${response.UserName} via
    the ${response.AccessKeyLastUsed.ServiceName} service on
    ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
    `);
  }

  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetAccessKeyLastUsado na referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getAccessKeyLastUsed(
  { AccessKeyId: "ACCESS_KEY_ID" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
```

```
    }
  }
);
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetAccessKeyLastUsado na referência AWS SDK for JavaScript da API.

`GetAccountPasswordPolicy`

O código de exemplo a seguir mostra como usar `GetAccountPasswordPolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha a política de senha da conta.

```
import {
  GetAccountPasswordPolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const getAccountPasswordPolicy = async () => {
  const command = new GetAccountPasswordPolicyCommand({});

  const response = await client.send(command);
  console.log(response.PasswordPolicy);
  return response;
};
```

- Para obter detalhes da API, consulte GetAccountPasswordPolicya Referência AWS SDK for JavaScript da API.

`GetPolicy`

O código de exemplo a seguir mostra como usar `GetPolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha a política.

```
import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const getPolicy = (policyArn) => {
  const command = new GetPolicyCommand({
    PolicyArn: policyArn,
  });

  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetPolicya Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetPolicya Referência AWS SDK for JavaScript da API.

`GetRole`

O código de exemplo a seguir mostra como usar `GetRole`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha a função.

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const getRole = (roleName) => {
  const command = new GetRoleCommand({
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetRolea Referência AWS SDK for JavaScript da API.

`GetServerCertificate`

O código de exemplo a seguir mostra como usar `GetServerCertificate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha um certificado do servidor.

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
```

```
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetServerCertificado na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetServerCertificado na Referência AWS SDK for JavaScript da API.

`GetServiceLinkedRoleDeletionStatus`

O código de exemplo a seguir mostra como usar `GetServiceLinkedRoleDeletionStatus`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetServiceLinkedRoleDeletionStatusa Referência AWS SDK for JavaScript da API.

`ListAccessKeys`

O código de exemplo a seguir mostra como usar `ListAccessKeys`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste as chaves de acesso.

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
  const command = new ListAccessKeysCommand({
    MaxItems: 5,
    UserName: userName,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.AccessKeyMetadata?.length) {
    for (const key of response.AccessKeyMetadata) {
      yield key;
    }

    if (response.IsTruncated) {
```

```
      response = await client.send(
        new ListAccessKeysCommand({
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListAccessChaves na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};

iam.listAccessKeys(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
```

```
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListAccessChaves na Referência AWS SDK for JavaScript da API.

`ListAccountAliases`

O código de exemplo a seguir mostra como usar `ListAccountAliases`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os aliases de conta.

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 */
export async function* listAccountAliases() {
  const command = new ListAccountAliasesCommand({ MaxItems: 5 });

  let response = await client.send(command);

  while (response.AccountAliases?.length) {
    for (const alias of response.AccountAliases) {
      yield alias;
    }
```

```
    if (response.IsTruncated) {
      response = await client.send(
        new ListAccountAliasesCommand({
          Marker: response.Marker,
          MaxItems: 5,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListAccountAliases na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListAccountAliases na Referência AWS SDK for JavaScript da API.

`ListAttachedRolePolicies`

O código de exemplo a seguir mostra como usar `ListAttachedRolePolicies`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Lista as políticas que estão anexadas a uma função.

```
import {
  ListAttachedRolePoliciesCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
  const command = new ListAttachedRolePoliciesCommand({
    RoleName: roleName,
  });

  let response = await client.send(command);
```

```
  while (response.AttachedPolicies?.length) {
    for (const policy of response.AttachedPolicies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAttachedRolePoliciesCommand({
          RoleName: roleName,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Para obter detalhes da API, consulte ListAttachedRolePoliciesa Referência AWS SDK for JavaScript da API.

`ListGroups`

O código de exemplo a seguir mostra como usar `ListGroups`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os grupos.

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
```

```
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 */
export async function* listGroups() {
  const command = new ListGroupsCommand({
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.Groups?.length) {
    for (const group of response.Groups) {
      yield group;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListGroupsCommand({
          Marker: response.Marker,
          MaxItems: 10,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Para obter detalhes da API, consulte ListGroupsa Referência AWS SDK for JavaScript da API.

`ListPolicies`

O código de exemplo a seguir mostra como usar `ListPolicies`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste as políticas.

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listPolicies() {
  const command = new ListPoliciesCommand({
    MaxItems: 10,
    OnlyAttached: false,
    // List only the customer managed policies in your Amazon Web Services account.
    Scope: "Local",
  });

  let response = await client.send(command);

  while (response.Policies?.length) {
    for (const policy of response.Policies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListPoliciesCommand({
          Marker: response.Marker,
          MaxItems: 10,
          OnlyAttached: false,
          Scope: "Local",
        }),
      );
    } else {
      break;
    }
  }
}
```

- Para obter detalhes da API, consulte ListPoliciesa Referência AWS SDK for JavaScript da API.

`ListRolePolicies`

O código de exemplo a seguir mostra como usar `ListRolePolicies`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste as políticas.

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 * @param {string} roleName
 */
export async function* listRolePolicies(roleName) {
  const command = new ListRolePoliciesCommand({
    RoleName: roleName,
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.PolicyNames?.length) {
    for (const policyName of response.PolicyNames) {
      yield policyName;
    }

    if (response.IsTruncated) {
```

```
      response = await client.send(
        new ListRolePoliciesCommand({
          RoleName: roleName,
          MaxItems: 10,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Para obter detalhes da API, consulte ListRolePolíticas na Referência AWS SDK for JavaScript da API.

`ListRoles`

O código de exemplo a seguir mostra como usar `ListRoles`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os perfis.

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
```

```
 */
export async function* listRoles() {
  const command = new ListRolesCommand({
    MaxItems: 10,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.Roles?.length) {
    for (const role of response.Roles) {
      yield role;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolesCommand({
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Para obter detalhes da API, consulte ListRolesa Referência AWS SDK for JavaScript da API.

`ListSAMLProviders`

O código de exemplo a seguir mostra como usar `ListSAMLProviders`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os provedores SAML.

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listSamlProviders = async () => {
  const command = new ListSAMLProvidersCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte ListSAMLProvider na Referência da API AWS SDK for JavaScript .

`ListServerCertificates`

O código de exemplo a seguir mostra como usar `ListServerCertificates`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os certificados.

```
import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
```

```
 *
 */
export async function* listServerCertificates() {
  const command = new ListServerCertificatesCommand({});
  let response = await client.send(command);

  while (response.ServerCertificateMetadataList?.length) {
    for await (const cert of response.ServerCertificateMetadataList) {
      yield cert;
    }

    if (response.IsTruncated) {
      response = await client.send(new ListServerCertificatesCommand({}));
    } else {
      break;
    }
  }
}
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListServerCertificados na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListServerCertificados na Referência AWS SDK for JavaScript da API.

`ListUsers`

O código de exemplo a seguir mostra como usar `ListUsers`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os usuários.

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listUsers = async () => {
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);
  response.Users?.forEach(({ UserName, CreateDate }) => {
    console.log(`${UserName} created on: ${CreateDate}`);
  });
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.

- Para obter detalhes da API, consulte ListUsersa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
      console.log("User " + user.UserName + " created", user.CreateDate);
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListUsersa Referência AWS SDK for JavaScript da API.

`PutRolePolicy`

O código de exemplo a seguir mostra como usar `PutRolePolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::some-test-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte PutRolePolítica na Referência AWS SDK for JavaScript da API.

`UpdateAccessKey`

O código de exemplo a seguir mostra como usar `UpdateAccessKey`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Atualize a chave de acesso.

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const updateAccessKey = (userName, accessKeyId) => {
  const command = new UpdateAccessKeyCommand({
    AccessKeyId: accessKeyId,
    Status: StatusType.Inactive,
    UserName: userName,
  });

  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte UpdateAccessChave na referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
```

```
  UserName: "USER_NAME",
};

iam.updateAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte UpdateAccessChave na referência AWS SDK for JavaScript da API.

`UpdateServerCertificate`

O código de exemplo a seguir mostra como usar `UpdateServerCertificate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Atualize um certificado do servidor.

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentName
 * @param {string} newName
 */
export const updateServerCertificate = (currentName, newName) => {
  const command = new UpdateServerCertificateCommand({
```

```
    ServerCertificateName: currentName,
    NewServerCertificateName: newName,
  });

  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte UpdateServerCertificado na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

iam.updateServerCertificate(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte UpdateServerCertificado na Referência AWS SDK for JavaScript da API.

`UpdateUser`

O código de exemplo a seguir mostra como usar `UpdateUser`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Atualize o usuário.

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentUserName
 * @param {string} newUserName
 */
export const updateUser = (currentUserName, newUserName) => {
  const command = new UpdateUserCommand({
    UserName: currentUserName,
    NewUserName: newUserName,
  });

  return client.send(command);
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte UpdateUsera Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

iam.updateUser(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte UpdateUsera Referência AWS SDK for JavaScript da API.

`UploadServerCertificate`

O código de exemplo a seguir mostra como usar `UploadServerCertificate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
 the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }

    throw err;
  }
};
```

```
/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

  return client.send(command);
};
```

- Para obter detalhes da API, consulte UploadServerCertificado na Referência AWS SDK for JavaScript da API.

Cenários

Criar e gerenciar um serviço resiliente

O exemplo de código a seguir mostra como criar um serviço web com balanceamento de carga que retorna recomendações de livros, filmes e músicas. O exemplo mostra como o serviço responde a falhas e como é possível reestruturá-lo para gerar mais resiliência em caso de falhas.

- Use um grupo do Amazon EC2 Auto Scaling para criar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) com base em um modelo de execução e para manter o número de instâncias em um intervalo especificado.
- Gerencie e distribua solicitações HTTP com o Elastic Load Balancing.
- Monitore a integridade das instâncias em um grupo do Auto Scaling e encaminhe solicitações somente para instâncias íntegras.
- Execute um servidor Web Python em cada instância do EC2 para lidar com solicitações HTTP. O servidor Web responde com recomendações e verificações de integridade.
- Simule um serviço de recomendação com uma tabela do Amazon DynamoDB.
- Controle a resposta do servidor web às solicitações e verificações de saúde atualizando AWS Systems Manager os parâmetros.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Execute o cenário interativo em um prompt de comando.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

Criar etapas para implantar todos os recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
```

```
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
```

```
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
```

```
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
```

```
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
```

```
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
```

```
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
```

```
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
```

```
        MaxSize: 3,
      }),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
   */
  (state) =>
    MESSAGES.createdAutoScalingGroup
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
      .replace(
        "${AVAILABILITY_ZONE_NAMES}",
        state.availabilityZoneNames.join(", "),
      ),
),
new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
  type: "confirm",
}),
new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
new ScenarioAction("getVpc", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
```

```
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
```

```
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
```

```
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
```

```
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
```

```
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
```

```
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

Criar etapas para executar a demonstração.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
```

```
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
```

```
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
```

```
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
```

```
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
```

```
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
```

```
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
 state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
```

```
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
```

```
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
```

```
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
```

```
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
```

```
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Criar etapas para destruir todos os recursos.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
```

```
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
```

```
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
```

```
      }),
    );
  }
}),
new ScenarioOutput("deletePolicyResult", (state) => {
  if (state.deletePolicyError) {
    console.error(state.deletePolicyError);
    return MESSAGES.deletePolicyError.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  } else {
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }
}),
new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.removeRoleFromInstanceProfileError = e;
  }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
```

```
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
```

```
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
```

```
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
```

```
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
```

```
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
```

```
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGrupos
  - CreateAutoScalingGroup
  - CreateInstancePerfil
  - CreateLaunchModelo
  - CreateListener
  - CreateLoadBalanceador
  - CreateTargetGrupo
  - DeleteAutoScalingGroup
  - DeleteInstancePerfil
  - DeleteLaunchModelo
  - DeleteLoadBalanceador
  - DeleteTargetGrupo
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZonas
  - DescribeIamInstanceProfileAssociações
  - DescribeInstances
  - DescribeLoadBalanceadores
  - DescribeSubnets
  - DescribeTargetGrupos
  - DescribeTargetHealth
  - DescribeVpcs

- RebootInstances
- ReplaceIamInstanceProfileAssociação
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Criar um usuário e assumir uma função

O exemplo de código a seguir mostra como criar um usuário e assumir um perfil.

> **Warning**
>
> Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como AWS IAM Identity Center.

- Crie um usuário sem permissões.
- Crie uma função que conceda permissão para listar os buckets do Amazon S3 para a conta.
- Adicione uma política para permitir que o usuário assuma a função.
- Assuma o perfil e liste buckets do S3 usando credenciais temporárias, depois limpe os recursos.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no repositório de exemplos de código da AWS.

Crie um usuário e um perfil do IAM que conceda permissão para listar os buckets do Amazon S3. O usuário só tem direitos para assumir a função. Após assumir a função, use credenciais temporárias para listar os buckets para a conta.

```
import {
  CreateUserCommand,
  GetUserCommand,
  CreateAccessKeyCommand,
```

```
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
```

```
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
      return User;
    } else {
      throw caught;
    }
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
 STS).
  // It's not best practice to use access keys. For more information, see https://
aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;
```

```
  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
      return await listBuckets(s3Client);
    } catch (err) {
      if (err instanceof Error && err.name === "InvalidAccessKeyId") {
        throw err;
      }
    }
  });

  // Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
  // is thrown while the user and access keys are still stabilizing.
  const { Role } = await retry(
    {
      intervalInMs: 2000,
      maxRetries: 60,
    },
    () =>
      iamClient.send(
        new CreateRoleCommand({
          AssumeRolePolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
              {
                Effect: "Allow",
                Principal: {
                  // Allow the previously created user to assume this role.
                  AWS: User.Arn,
                },
                Action: "sts:AssumeRole",
              },
            ],
          }),
```

```
        RoleName: roleName,
      }),
    ),
);

if (!Role) {
  throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["s3:ListAllMyBuckets"],
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  }),
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
```

```
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);
```

```
  await iamClient.send(
    new DeletePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
    }),
  );

  await iamClient.send(
    new DeleteRoleCommand({
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - AttachRolePolítica
  - CreateAccessChave
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessChave
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolítica
  - DetachRolePolítica
  - PutUserPolítica

Exemplos do Kinesis usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Kinesis.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Exemplos sem servidor

Exemplos sem servidor

Invocar uma função do Lambda em um trigger do Kinesis

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de um stream do Kinesis. A função recupera a carga útil do Kinesis, decodifica do Base64 e registra o conteúdo do registro em log.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do Kinesis com o uso do Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Consumindo um evento do Kinesis com o uso do Lambda. 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;
}
```

Relatando falhas de itens em lote para funções do Lambda com um trigger do Kinesis

O exemplo de código a seguir mostra 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.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre 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;
}
```

Relatando falhas de itens em lote do Kinesis com o uso do Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
```

```
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Exemplos de Lambda usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Lambda.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Lambda

Os exemplos de código a seguir mostram como começar a usar o Lambda.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
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 ListFunctionsa Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários
- Exemplos sem servidor

Ações

`CreateFunction`

O código de exemplo a seguir mostra como usar `CreateFunction`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
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 CreateFunctiona Referência AWS SDK for JavaScript da API.

`DeleteFunction`

O código de exemplo a seguir mostra como usar `DeleteFunction`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
/**
 * @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 DeleteFunctiona Referência AWS SDK for JavaScript da API.

`GetFunction`

O código de exemplo a seguir mostra como usar `GetFunction`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetFunctiona Referência AWS SDK for JavaScript da API.

`Invoke`

O código de exemplo a seguir mostra como usar `Invoke`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
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 .

`ListFunctions`

O código de exemplo a seguir mostra como usar `ListFunctions`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Para obter detalhes da API, consulte ListFunctionsa Referência AWS SDK for JavaScript da API.

`UpdateFunctionCode`

O código de exemplo a seguir mostra como usar `UpdateFunctionCode`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
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 da API, consulte UpdateFunctionCódigo na Referência AWS SDK for JavaScript da API.

`UpdateFunctionConfiguration`

O código de exemplo a seguir mostra como usar `UpdateFunctionConfiguration`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
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 UpdateFunctionConfiguração na Referência AWS SDK for JavaScript da API.

Cenários

Conceitos básicos de funções

O exemplo de código a seguir mostra como:

- Criar um perfil do IAM e uma função do Lambda e carregar 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.
- Listar as funções para sua conta e limpar os recursos.

Para obter mais informações, consulte Criar uma função do Lambda no console.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie uma função AWS Identity and Access Management (IAM) que conceda ao Lambda permissão para gravar em registros.

```
    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
  - UpdateFunctionCódigo
  - UpdateFunctionConfiguração

Exemplos sem servidor

Invocar uma função do Lambda em um trigger do Kinesis

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de um stream do Kinesis. A função recupera a carga útil do Kinesis, decodifica do Base64 e registra o conteúdo do registro em log.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do Kinesis com o uso do Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Consumindo um evento do Kinesis com o uso do Lambda. 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;
}
```

Invocar uma função do Lambda em um gatilho do DynamoDB

O exemplo de código a seguir mostra como implementar uma função Lambda que recebe um evento acionado pelo recebimento de registros de um stream do DynamoDB. A função recupera a carga útil do DynamoDB e registra em log o conteúdo do registro.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do DynamoDB com o uso do Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consumindo um evento do DynamoDB com o uso do Lambda. 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)}`);
};
```

Invocar uma função Lambda a partir de um gatilho do Amazon DocumentDB

O exemplo de código a seguir mostra como implementar uma função Lambda que recebe um evento acionado pelo recebimento de registros de um stream de alterações do DocumentDB. A função recupera a carga do DocumentDB e registra o conteúdo do registro.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do Amazon DocumentDB com o uso do Lambda. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

Invocar uma função do Lambda em um acionador do Amazon S3

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo upload de um objeto para 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 o tipo de conteúdo do objeto.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do S3 com o uso do JavaScript Lambda.

```
// 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);
    }
};
```

Consumindo um evento do S3 com o uso do TypeScript Lambda.

```
// 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);
  }
};
```

Invocar uma função do Lambda em um acionador do Amazon SNS

O exemplo de código a seguir mostra 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.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do SNS com o JavaScript Lambda usando.

```
// 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;
  }
}
```

Consumindo um evento do SNS com o TypeScript Lambda usando.

```
// 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;
  }
}
```

Invocar uma função do Lambda em um trigger do Amazon SQS

O exemplo de código a seguir mostra 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.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento SQS com o JavaScript Lambda usando.

```
// 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;
  }
}
```

Consumindo um evento SQS com o TypeScript Lambda usando.

```
// 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;
```

```
  }
}
```

Relatando falhas de itens em lote para funções do Lambda com um trigger do Kinesis

O exemplo de código a seguir mostra 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.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre 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;
}
```

Relatando falhas de itens em lote do Kinesis com o uso do Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
```

```
            Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB

O exemplo de código a seguir mostra como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um stream do DynamoDB. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatando falhas de itens em lote do DynamoDB com o uso do Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";
```

```
  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Relatando falhas de itens em lote do DynamoDB com o uso do Lambda. TypeScript

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

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

    const batchItemsFailures: DynamoDBBatchItemFailure[] = []
    let curRecordSequenceNumber

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

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

    return batchItemsFailures
}
```

Relatar falhas de itens em lote para funções do Lambda com um trigger do Amazon SQS

O exemplo de código a seguir mostra 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.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatando falhas de itens em lote do SQS com o uso do JavaScript Lambda.

```
// 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}`);
}
```

Relatando falhas de itens em lote do SQS com o uso do TypeScript Lambda.

```
// 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}`);
}
```

Exemplos do Amazon Personalize usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Personalize.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Ações

`CreateBatchInferenceJob`

O código de exemplo a seguir mostra como usar `CreateBatchInferenceJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchInferenceJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch inference job's parameters.

export const createBatchInferenceJobParam = {
  jobName: 'JOB_NAME',
  jobInput: {         /* required */
    s3DataSource: {   /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
```

```
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional integer*/
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchInferenceJobCommand(createBatchInferenceJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateBatchInferenceJoba Referência AWS SDK for JavaScript da API.

`CreateBatchSegmentJob`

O código de exemplo a seguir mostra como usar `CreateBatchSegmentJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});
```

```
// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: 'NAME',
  jobInput: {         /* required */
    s3DataSource: {   /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchSegmentJobCommand(createBatchSegmentJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateBatchSegmentJoba Referência AWS SDK for JavaScript da API.

`CreateCampaign`

O código de exemplo a seguir mostra como usar `CreateCampaign`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.

import { CreateCampaignCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the campaign's parameters.
export const createCampaignParam = {
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  name: 'NAME',  /* required */
  minProvisionedTPS: 1    /* optional integer */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateCampaignCommand(createCampaignParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateCampaigna Referência AWS SDK for JavaScript da API.

`CreateDataset`

O código de exemplo a seguir mostra como usar `CreateDataset`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset's parameters.
export const createDatasetParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  datasetType: 'DATASET_TYPE', /* required */
  name: 'NAME', /* required */
  schemaArn: 'SCHEMA_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetCommand(createDatasetParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateDataseta Referência AWS SDK for JavaScript da API.

`CreateDatasetExportJob`

O código de exemplo a seguir mostra como usar `CreateDatasetExportJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetExportJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the export job parameters.
export const datasetExportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  jobOutput: {
    s3DataDestination: {
        path: 'S3_DESTINATION_PATH' /* required */
        //kmsKeyArn: 'ARN'  /* include if your bucket uses AWS KMS for encryption
    }
  },
  jobName: 'NAME',/* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetExportJobCommand(datasetExportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
```

```
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateDatasetExportJoba Referência AWS SDK for JavaScript da API.

`CreateDatasetGroup`

O código de exemplo a seguir mostra como usar `CreateDatasetGroup`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.

import { CreateDatasetGroupCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset group parameters.
export const createDatasetGroupParam = {
  name: "NAME" /* required */,
};

export const run = async (createDatasetGroupParam) => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetGroupCommand(createDatasetGroupParam),
    );
    console.log("Success", response);
```

```
    return "Run successfully"; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run(createDatasetGroupParam);
```

Criar um grupo de conjunto de dados de domínio.

```
// Get service clients module and commands using ES6 syntax.
import {  CreateDatasetGroupCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the domain dataset group parameters.
export const domainDatasetGroupParams = {
  name: 'NAME',  /* required */
  domain: 'DOMAIN'   /* required for a domain dsg, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetGroupCommand(domainDatasetGroupParams));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateDatasetGrupo na Referência AWS SDK for JavaScript da API.

`CreateDatasetImportJob`

O código de exemplo a seguir mostra como usar `CreateDatasetImportJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import {CreateDatasetImportJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset import job parameters.
export const datasetImportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  dataSource: {  /* required */
    dataLocation: 'S3_PATH'
  },
  jobName: 'NAME',/* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetImportJobCommand(datasetImportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateDatasetImportJoba Referência AWS SDK for JavaScript da API.

`CreateEventTracker`

O código de exemplo a seguir mostra como usar `CreateEventTracker`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateEventTrackerCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the event tracker's parameters.
export const createEventTrackerParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateEventTrackerCommand(createEventTrackerParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateEventRastreador na Referência AWS SDK for JavaScript da API.

`CreateFilter`

O código de exemplo a seguir mostra como usar `CreateFilter`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateFilterCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the filter's parameters.
export const createFilterParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
  filterExpression: 'FILTER_EXPRESSION' /*required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateFilterCommand(createFilterParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateFiltera Referência AWS SDK for JavaScript da API.

`CreateRecommender`

O código de exemplo a seguir mostra como usar `CreateRecommender`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateRecommenderCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the recommender's parameters.
export const createRecommenderParam = {
  name: 'NAME', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  datasetGroupArn: 'DATASET_GROUP_ARN'  /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateRecommenderCommand(createRecommenderParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateRecommendera Referência AWS SDK for JavaScript da API.

`CreateSchema`

O código de exemplo a seguir mostra como usar `CreateSchema`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema /* required */
};

export const run = async () => {
  try {
```

```
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Crie um esquema com um domínio.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema, /* required */
  domain: 'DOMAIN'   /* required for a domain dataset group, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
};

export const run = async () => {
  try {
```

```
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createDomainSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateSchemaa Referência AWS SDK for JavaScript da API.

`CreateSolution`

O código de exemplo a seguir mostra como usar `CreateSolution`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution parameters.
export const createSolutionParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  name: 'NAME' /* required */
}

export const run = async () => {
```

```
  try {
    const response = await personalizeClient.send(new
 CreateSolutionCommand(createSolutionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateSolutiona Referência AWS SDK for JavaScript da API.

`CreateSolutionVersion`

O código de exemplo a seguir mostra como usar `CreateSolutionVersion`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionVersionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution version parameters.
export const solutionVersionParam = {
  solutionArn: 'SOLUTION_ARN' /* required */
}

export const run = async () => {
  try {
```

```
    const response = await personalizeClient.send(new
 CreateSolutionVersionCommand(solutionVersionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateSolutionVersão na Referência AWS SDK for JavaScript da API.

Exemplos de eventos do Amazon Personalize usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Personalize Events.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`PutEvents`

O código de exemplo a seguir mostra como usar `PutEvents`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
 Replace it with your sentAt timestamp in UNIX format.

// Set put events parameters.
var putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
```

```
  }
};
run();
```

- Para obter detalhes da API, consulte PutEventsa Referência AWS SDK for JavaScript da API.

`PutItems`

O código de exemplo a seguir mostra como usar `PutItems`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { PutItemsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put items parameters. For string properties and values, use the \
 character to escape quotes.
var putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /*  required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
 "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
```

```
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte PutItemsa Referência AWS SDK for JavaScript da API.

`PutUsers`

O código de exemplo a seguir mostra como usar `PutUsers`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { PutUsersCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put users parameters. For string properties and values, use the \
 character to escape quotes.
var putUsersParam = {
  datasetArn: "DATASET_ARN",
  users: [
    {
      userId: "USER_ID",
      properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
    },
  ],
};
```

```
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte PutUsersa Referência AWS SDK for JavaScript da API.

Exemplos do Amazon Personalize Runtime usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Personalize Runtime.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`GetPersonalizedRanking`

O código de exemplo a seguir mostra como usar `GetPersonalizedRanking`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { GetPersonalizedRankingCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the ranking request parameters.
export const getPersonalizedRankingParam = {
  campaignArn: "CAMPAIGN_ARN", /* required */
  userId: 'USER_ID',      /* required */
  inputList: ["ITEM_ID_1", "ITEM_ID_2", "ITEM_ID_3", "ITEM_ID_4"]
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetPersonalizedRankingCommand(getPersonalizedRankingParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte GetPersonalizedClassificação na referência AWS SDK for JavaScript da API.

`GetRecommendations`

O código de exemplo a seguir mostra como usar `GetRecommendations`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";

import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',      /* required */
  numResults: 15    /* optional */
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Obtenha recomendações com um filtro (grupo de conjunto de dados personalizados).

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  recommenderArn: 'RECOMMENDER_ARN', /* required */
  userId: 'USER_ID',       /* required */
  numResults: 15    /* optional */
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Obtenha recomendações filtradas de um recomendador criado em um grupo de conjunto de dados de domínio.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here:
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set recommendation request parameters.
export const getRecommendationsParam = {
```

```
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',      /* required */
  numResults: 15,    /* optional */
  filterArn: 'FILTER_ARN',   /* required to filter recommendations */
  filterValues: {
    "PROPERTY": "\"VALUE\""  /* Only required if your filter has a placeholder
 parameter */
  }
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte GetRecommendationsa Referência AWS SDK for JavaScript da API.

Exemplos do Amazon Pinpoint usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Pinpoint.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`SendMessages`

O código de exemplo a seguir mostra como usar `SendMessages`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
export const pinClient = new PinpointClient({ region: REGION });
```

Envie uma mensagem de e-mail.

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "./libs/pinClient.js";

// The FromAddress must be verified in SES.
const fromAddress = "FROM_ADDRESS";
const toAddress = "TO_ADDRESS";
const projectId = "PINPOINT_PROJECT_ID";

// The subject line of the email.
var subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
----------------------------------------------------
```

```
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
 using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding for the subject line and message body of the email.
var charset = "UTF-8";

const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
```

```
        },
      },
    },
  },
};

const run = async () => {
  try {
    const { MessageResponse } = await pinClient.send(
      new SendMessagesCommand(params),
    );

    if (!MessageResponse) {
      throw new Error("No message response.");
    }

    if (!MessageResponse.Result) {
      throw new Error("No message result.");
    }

    const recipientResult = MessageResponse.Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    } else {
      console.log(recipientResult.MessageId);
    }
  } catch (err) {
    console.log(err.message);
  }
};

run();
```

Envie uma mensagem SMS.

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "./libs/pinClient.js";

/* The phone number or short code to send the message from. The phone number
```

```
 or short code that you specify has to be associated with your Amazon Pinpoint
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number.  For best results, you should specify the phone
 number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderID";

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
```

```
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"],
    );
  } catch (err) {
    console.log(err);
  }
};
run();
```

- Para obter detalhes da API, consulte SendMessagesa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de e-mail.

```
"use strict";

const AWS = require("aws-sdk");

// The AWS Region that you want to use to send the email. For a list of
```

```
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
var toAddress = "recipient@example.com";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https:\/\/aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
```

```
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: appId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: senderAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

//Try to send the email.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
```

```
    console.log(err.message);
  } else {
    console.log(
      "Email sent! Message ID: ",
      data["MessageResponse"]["Result"][toAddress]["MessageId"]
    );
  }
});
```

Envie uma mensagem SMS.

```
"use strict";

var AWS = require("aws-sdk");

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";

// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
var originationNumber = "+12065550199";

// The recipient's phone number.  For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";
```

```
// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderID";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};
```

```
//Try to send the message.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
    // Otherwise, show the unique ID for the message.
  } else {
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  }
});
```

- Para obter detalhes da API, consulte SendMessagesa Referência AWS SDK for JavaScript da API.

Exemplos do Amazon Redshift usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Redshift.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`CreateCluster`

O código de exemplo a seguir mostra como usar `CreateCluster`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Crie o cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least
 one uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
 your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
 cluster subnet group to be associated with this cluster. Defaults to 'default' if
 not specified.
```

```
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte CreateClustera Referência AWS SDK for JavaScript da API.

`DeleteCluster`

O código de exemplo a seguir mostra como usar `DeleteCluster`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Crie o cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte DeleteClustera Referência AWS SDK for JavaScript da API.

`DescribeClusters`

O código de exemplo a seguir mostra como usar `DescribeClusters`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Descreva os clusters.

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte DescribeClusters a Referência AWS SDK for JavaScript da API.

`ModifyCluster`

O código de exemplo a seguir mostra como usar `ModifyCluster`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Modificar um cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter detalhes da API, consulte ModifyClustera Referência AWS SDK for JavaScript da API.

Exemplos do Amazon S3 usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon S3.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Amazon S3

O exemplo de código a seguir mostra como começar a usar o Amazon S3.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new S3Client({});

export const helloS3 = async () => {
  const command = new ListBucketsCommand({});
```

```
  const { Buckets } = await client.send(command);
  console.log("Buckets: ");
  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
  return Buckets;
};
```

- Para obter detalhes da API, consulte ListBucketsa Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários
- Exemplos sem servidor

Ações

`CopyObject`

O código de exemplo a seguir mostra como usar `CopyObject`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Copie o objeto.

```
import { S3Client, CopyObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CopyObjectCommand({
    CopySource: "SOURCE_BUCKET/SOURCE_OBJECT_KEY",
    Bucket: "DESTINATION_BUCKET",
```

```
    Key: "NEW_OBJECT_KEY",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte CopyObjecta Referência AWS SDK for JavaScript da API.

`CreateBucket`

O código de exemplo a seguir mostra como usar `CreateBucket`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie o bucket.

```
import { CreateBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CreateBucketCommand({
    // The name of the bucket. Bucket names are unique and have several other
 constraints.
    // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
bucketnamingrules.html
    Bucket: "bucket-name",
  });

  try {
```

```
    const { Location } = await client.send(command);
    console.log(`Bucket created with location ${Location}`);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateBucketa Referência AWS SDK for JavaScript da API.

`DeleteBucket`

O código de exemplo a seguir mostra como usar `DeleteBucket`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Excluir o bucket.

```
import { DeleteBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Delete a bucket.
export const main = async () => {
  const command = new DeleteBucketCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
```

```
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteBucketa Referência AWS SDK for JavaScript da API.

`DeleteBucketPolicy`

O código de exemplo a seguir mostra como usar `DeleteBucketPolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua a política de bucket.

```
import { DeleteBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// This will remove the policy from the bucket.
export const main = async () => {
  const command = new DeleteBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteBucketPolítica na Referência AWS SDK for JavaScript da API.

`DeleteBucketWebsite`

O código de exemplo a seguir mostra como usar `DeleteBucketWebsite`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua a configuração de site do bucket.

```
import { DeleteBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Disable static website hosting on the bucket.
export const main = async () => {
  const command = new DeleteBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteBucketSite na Referência AWS SDK for JavaScript da API.

`DeleteObject`

O código de exemplo a seguir mostra como usar `DeleteObject`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua um objeto.

```
import { DeleteObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectCommand({
    Bucket: "test-bucket",
    Key: "test-key.txt",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DeleteObjecta Referência AWS SDK for JavaScript da API.

`DeleteObjects`

O código de exemplo a seguir mostra como usar `DeleteObjects`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Exclua vários objetos.

```
import { DeleteObjectsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectsCommand({
    Bucket: "test-bucket",
    Delete: {
      Objects: [{ Key: "object1.txt" }, { Key: "object2.txt" }],
    },
  });

  try {
    const { Deleted } = await client.send(command);
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted
 objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DeleteObjectsa Referência AWS SDK for JavaScript da API.

`GetBucketAcl`

O código de exemplo a seguir mostra como usar `GetBucketAcl`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha as permissões de ACL.

```
import { GetBucketAclCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketAclCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetBucketAcl em Referência AWS SDK for JavaScript da API.

`GetBucketCors`

O código de exemplo a seguir mostra como usar `GetBucketCors`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha a política de CORS para o bucket.

```
import { GetBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketCorsCommand({
    Bucket: "test-bucket",
  });

  try {
    const { CORSRules } = await client.send(command);
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `\n${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders.join(" ")}`,
        `\nAllowedMethods: ${cr.AllowedMethods.join(" ")}`,
        `\nAllowedOrigins: ${cr.AllowedOrigins.join(" ")}`,
        `\nExposeHeaders: ${cr.ExposeHeaders.join(" ")}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetBucketCors em Referência AWS SDK for JavaScript da API.

`GetBucketPolicy`

O código de exemplo a seguir mostra como usar `GetBucketPolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha a política de bucket.

```
import { GetBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const { Policy } = await client.send(command);
    console.log(JSON.parse(Policy));
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetBucketPolítica na Referência AWS SDK for JavaScript da API.

`GetBucketWebsite`

O código de exemplo a seguir mostra como usar `GetBucketWebsite`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha a configuração do site.

```
import { GetBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const { ErrorDocument, IndexDocument } = await client.send(command);
    console.log(
      `Your bucket is set up to host a website. It has an error document:`,
      `${ErrorDocument.Key}, and an index document: ${IndexDocument.Suffix}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte GetBucketSite na Referência AWS SDK for JavaScript da API.

`GetObject`

O código de exemplo a seguir mostra como usar `GetObject`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Baixe o objeto.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
  });

  try {
    const response = await client.send(command);
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetObjecta Referência AWS SDK for JavaScript da API.

`GetObjectLockConfiguration`

O código de exemplo a seguir mostra como usar `GetObjectLockConfiguration`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import {
  GetObjectLockConfigurationCommand,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new GetObjectLockConfigurationCommand({
    Bucket: bucketName,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
  });

  try {
    const { ObjectLockConfiguration } = await client.send(command);
    console.log(`Object Lock Configuration: ${ObjectLockConfiguration}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

- Para obter detalhes da API, consulte GetObjectLockConfigurationa Referência AWS SDK for JavaScript da API.

`GetObjectRetention`

O código de exemplo a seguir mostra como usar `GetObjectRetention`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import { GetObjectRetentionCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const { Retention } = await client.send(command);
    console.log(`Object Retention Settings: ${Retention.Status}`);
  } catch (err) {
    console.error(err);
  }
};
```

```
// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Para obter detalhes da API, consulte GetObjectRetenção na Referência AWS SDK for JavaScript da API.

`ListBuckets`

O código de exemplo a seguir mostra como usar `ListBuckets`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste os buckets.

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListBucketsCommand({});

  try {
    const { Owner, Buckets } = await client.send(command);
    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`,
    );
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (err) {
    console.error(err);
  }
```

```
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListBucketsa Referência AWS SDK for JavaScript da API.

`ListObjectsV2`

O código de exemplo a seguir mostra como usar `ListObjectsV2`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Liste todos os objetos no bucket. Se houver mais de um objeto, IsTruncated NextContinuationToken ele será usado para iterar a lista completa.

```
import {
  S3Client,
  // This command supersedes the ListObjectsCommand and is the recommended way to
 list objects.
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListObjectsV2Command({
    Bucket: "my-bucket",
    // The default and maximum number of keys returned is 1000. This limits it to
    // one for demonstration purposes.
    MaxKeys: 1,
  });

  try {
    let isTruncated = true;
```

```
    console.log("Your bucket contains the following objects:\n");
    let contents = "";

    while (isTruncated) {
      const { Contents, IsTruncated, NextContinuationToken } =
        await client.send(command);
      const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
      contents += contentsList + "\n";
      isTruncated = IsTruncated;
      command.input.ContinuationToken = NextContinuationToken;
    }
    console.log(contents);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte ListObjectsV2 na Referência AWS SDK for JavaScript da API.

`PutBucketAcl`

O código de exemplo a seguir mostra como usar `PutBucketAcl`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Coloque a ACL do bucket.

```
import { PutBucketAclCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Most Amazon S3 use cases don't require the use of access control lists (ACLs).
// We recommend that you disable ACLs, except in unusual circumstances where
```

```
// you need to control access for each object individually.
// Consider a policy instead. For more information see https://docs.aws.amazon.com/
AmazonS3/latest/userguide/bucket-policies.html.
export const main = async () => {
  // Grant a user READ access to a bucket.
  const command = new PutBucketAclCommand({
    Bucket: "test-bucket",
    AccessControlPolicy: {
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
 AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
            ID: "canonical-id-1",
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "FULL_CONTROL",
        },
      ],
      Owner: {
        ID: "canonical-id-2",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutBucketAcl em Referência AWS SDK for JavaScript da API.

`PutBucketCors`

O código de exemplo a seguir mostra como usar `PutBucketCors`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Adicione uma regra de CORS.

```
import { PutBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// By default, Amazon S3 doesn't allow cross-origin requests. Use this command
// to explicitly allow cross-origin requests.
export const main = async () => {
  const command = new PutBucketCorsCommand({
    Bucket: "test-bucket",
    CORSConfiguration: {
      CORSRules: [
        {
          // Allow all headers to be sent to this bucket.
          AllowedHeaders: ["*"],
          // Allow only GET and PUT methods to be sent to this bucket.
          AllowedMethods: ["GET", "PUT"],
          // Allow only requests from the specified origin.
          AllowedOrigins: ["https://www.example.com"],
          // Allow the entity tag (ETag) header to be returned in the response. The
 ETag header
          // The entity tag represents a specific version of the object. The ETag
 reflects
          // changes only to the contents of an object, not its metadata.
          ExposeHeaders: ["ETag"],
          // How long the requesting browser should cache the preflight response.
 After
          // this time, the preflight request will have to be made again.
          MaxAgeSeconds: 3600,
        },
```

```
      ],
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutBucketCors em Referência AWS SDK for JavaScript da API.

`PutBucketPolicy`

O código de exemplo a seguir mostra como usar `PutBucketPolicy`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Adicione a política.

```
import { PutBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutBucketPolicyCommand({
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
```

```
          Sid: "AllowGetObject",
          // Allow this particular user to call GetObject on any object in this
 bucket.
          Effect: "Allow",
          Principal: {
            AWS: "arn:aws:iam::ACCOUNT-ID:user/USERNAME",
          },
          Action: "s3:GetObject",
          Resource: "arn:aws:s3:::BUCKET-NAME/*",
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: "BUCKET-NAME",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutBucketPolítica na Referência AWS SDK for JavaScript da API.

`PutBucketWebsite`

O código de exemplo a seguir mostra como usar `PutBucketWebsite`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Defina a configuração do site.

```
import { PutBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Set up a bucket as a static website.
// The bucket needs to be publicly accessible.
export const main = async () => {
  const command = new PutBucketWebsiteCommand({
    Bucket: "test-bucket",
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
        // A suffix that is appended to a request that is for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutBucketSite na Referência AWS SDK for JavaScript da API.

`PutObject`

O código de exemplo a seguir mostra como usar `PutObject`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Carregue o objeto.

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

const client = new S3Client({});

export const main = async () => {
  const command = new PutObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
    Body: "Hello S3!",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte PutObjecta Referência AWS SDK for JavaScript da API.

`PutObjectLegalHold`

O código de exemplo a seguir mostra como usar `PutObjectLegalHold`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import { PutObjectLegalHoldCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    LegalHold: {
      // Set the status to 'ON' to place a legal hold on the object.
      // Set the status to 'OFF' to remove the legal hold.
      Status: "ON",
    },
    // Optionally, you can provide additional parameters
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object legal hold status: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
```

```
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Para obter detalhes da API, consulte PutObjectLegalHolda Referência AWS SDK for JavaScript da API.

`PutObjectLockConfiguration`

O código de exemplo a seguir mostra como usar `PutObjectLockConfiguration`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Defina a configuração de Bloqueio de Objetos de um bucket.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import {
  PutObjectLockConfigurationCommand,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
```

```
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Lock Configuration updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

Defina o período de retenção padrão de um bucket.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import {
  PutObjectLockConfigurationCommand,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
```

```
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
      Rule: {
        DefaultRetention: {
          Mode: "GOVERNANCE",
          Years: 3,
        },
      },
    },
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Default Object Lock Configuration updated: ${response.
$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

- Para obter detalhes da API, consulte PutObjectLockConfigurationa Referência AWS SDK for JavaScript da API.

`PutObjectRetention`

O código de exemplo a seguir mostra como usar `PutObjectRetention`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import { PutObjectRetentionCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    BypassGovernanceRetention: false,
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    Retention: {
      Mode: "GOVERNANCE", // or "COMPLIANCE"
      RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Retention settings updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
```

```
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Para obter detalhes da API, consulte PutObjectRetenção na Referência AWS SDK for JavaScript da API.

Cenários

Criar um URL pré-assinado

O exemplo de código a seguir mostra como criar um URL pré-assinado para o Amazon S3 e fazer upload de um objeto.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie um URL pré-assinado para carregar um objeto em um bucket.

```
import https from "https";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
```

```
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

function put(url, data) {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          resolve(responseBody);
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
}

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
```

```
  const KEY = "example_file.txt";

  // There are two ways to generate a presigned URL.
  // 1. Use createPresignedUrl without the S3 client.
  // 2. Use getSignedUrl in conjunction with the S3 client and GetObjectCommand.
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    // After you get the presigned URL, you can provide your own file
    // data. Refer to put() above.
    console.log("Calling PUT using presigned URL without client");
    await put(noClientUrl, "Hello World");

    console.log("Calling PUT using presigned URL with client");
    await put(clientUrl, "Hello World");

    console.log("\nDone. Check your S3 console.");
  } catch (err) {
    console.error(err);
  }
};
```

Crie um URL pré-assinado para baixar um objeto de um bucket.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
```

```
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(new HttpRequest(url));
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new GetObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.jpg";

  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    console.log("Presigned URL without client");
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
```

```
    console.log(clientUrl);
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.

Criar uma página da web que oferece uma lista de objetos do Amazon S3

O exemplo de código a seguir mostra como listar objetos do Amazon S3 em uma página da web.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

O código a seguir é o componente relevante do React que faz chamadas para o AWS SDK. Uma versão executável do aplicativo contendo esse componente pode ser encontrada no link anterior GitHub .

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";

function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
```

```
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
 role to the pool,
      // and grant the role access to the 's3:GetObject' action.
      //
      // You'll also need to configure the CORS settings on the bucket to allow
 traffic from
      // this example site. Here's an example configuration that allows all origins.
 Don't
      // do this in production.
      //[
      //  {
      //    "AllowedHeaders": ["*"],
      //    "AllowedMethods": ["GET"],
      //    "AllowedOrigins": ["*"],
      //    "ExposeHeaders": [],
      //  },
      //]
      //
      credentials: fromCognitoIdentityPool({
        clientConfig: { region: "us-east-1" },
        identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
      }),
    });
    const command = new ListObjectsCommand({ Bucket: "bucket-name" });
    client.send(command).then(({ Contents }) => setObjects(Contents || []));
  }, []);

  return (
    <div className="App">
      {objects.map((o) => (
        <div key={o.ETag}>{o.Key}</div>
      ))}
    </div>
  );
}

export default App;
```

- Para obter detalhes da API, consulte ListObjectsa Referência AWS SDK for JavaScript da API.

Conceitos básicos de buckets e objetos

O exemplo de código a seguir mostra como:

- Criar um bucket e fazer upload de um arquivo para ele.
- Baixar um objeto de um bucket.
- Copiar um objeto em uma subpasta em um bucket.
- Listar os objetos em um bucket.
- Excluir os objetos do bucket e o bucket.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Primeiro, importe todos os módulos necessários.

```
// Used to check if currently running file is this file.
import { fileURLToPath } from "url";
import { readdirSync, readFileSync, writeFileSync } from "fs";

// Local helper utils.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

import {
  S3Client,
  CreateBucketCommand,
  PutObjectCommand,
  ListObjectsCommand,
  CopyObjectCommand,
  GetObjectCommand,
  DeleteObjectsCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
```

As importações anteriores fazem referência a alguns utilitários auxiliares. Esses utilitários são locais para o GitHub repositório vinculado no início desta seção. Para sua referência, consulte as implementações a seguir desses utilitários.

```
export const dirnameFromMetaUrl = (metaUrl) =>
  fileURLToPath(new URL(".", metaUrl));

import { select, input, confirm, checkbox } from "@inquirer/prompts";

export class Prompter {
  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  select(options) {
    return select(options);
  }

  /**
   * @param {{ message: string }} options
   */
  input(options) {
    return input(options);
  }

  /**
   * @param {string} prompt
   */
  checkContinue = async (prompt = "") => {
    const prefix = prompt && prompt + " ";
    let ok = await this.confirm({
      message: `${prefix}Continue?`,
    });
    if (!ok) throw new Error("Exiting...");
  };

  /**
   * @param {{ message: string }} options
   */
  confirm(options) {
    return confirm(options);
  }
```

```
  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  checkbox(options) {
    return checkbox(options);
  }
}

export const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};
```

Os objetos no S3 são armazenados em “buckets”. Vamos definir uma função para criar um bucket.

```
export const createBucket = async () => {
  const bucketName = await prompter.input({
    message: "Enter a bucket name. Bucket names must be globally unique:",
  });
  const command = new CreateBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log("Bucket created successfully.\n");
  return bucketName;
};
```

Os buckets contêm “objetos”. Essa função faz upload do conteúdo de um diretório para seu bucket como objetos.

```
export const uploadFilesToBucket = async ({ bucketName, folderPath }) => {
  console.log(`Uploading files from ${folderPath}\n`);
  const keys = readdirSync(folderPath);
  const files = keys.map((key) => {
    const filePath = `${folderPath}/${key}`;
    const fileContent = readFileSync(filePath);
    return {
      Key: key,
      Body: fileContent,
    };
  });
```

```
  for (let file of files) {
    await s3Client.send(
      new PutObjectCommand({
        Bucket: bucketName,
        Body: file.Body,
        Key: file.Key,
      }),
    );
    console.log(`${file.Key} uploaded successfully.`);
  }
};
```

Depois de fazer upload dos objetos, confira se eles foram carregados corretamente. Você pode usar ListObjects para isso. Você usará a propriedade “Key”, mas também há outras propriedades úteis na resposta.

```
export const listFilesInBucket = async ({ bucketName }) => {
  const command = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(command);
  const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
  console.log("\nHere's a list of files in the bucket:");
  console.log(contentsList + "\n");
};
```

Às vezes é necessário copiar um objeto de um bucket em outro. Use o CopyObject comando para isso.

```
export const copyFileFromBucket = async ({ destinationBucket }) => {
  const proceed = await prompter.confirm({
    message: "Would you like to copy an object from another bucket?",
  });

  if (!proceed) {
    return;
  } else {
    const copy = async () => {
      try {
        const sourceBucket = await prompter.input({
          message: "Enter source bucket name:",
```

```
        });
        const sourceKey = await prompter.input({
          message: "Enter source key:",
        });
        const destinationKey = await prompter.input({
          message: "Enter destination key:",
        });

        const command = new CopyObjectCommand({
          Bucket: destinationBucket,
          CopySource: `${sourceBucket}/${sourceKey}`,
          Key: destinationKey,
        });
        await s3Client.send(command);
        await copyFileFromBucket({ destinationBucket });
      } catch (err) {
        console.error(`Copy error.`);
        console.error(err);
        const retryAnswer = await prompter.confirm({ message: "Try again?" });
        if (retryAnswer) {
          await copy();
        }
      }
    };
    await copy();
  }
};
```

Não há um método de SDK para obter vários objetos de um bucket. Em vez disso, você criará uma lista de objetos para baixar e iterar sobre eles.

```
export const downloadFilesFromBucket = async ({ bucketName }) => {
  const { Contents } = await s3Client.send(
    new ListObjectsCommand({ Bucket: bucketName }),
  );
  const path = await prompter.input({
    message: "Enter destination path for files:",
  });

  for (let content of Contents) {
    const obj = await s3Client.send(
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
```

```
    );
    writeFileSync(
      `${path}/${content.Key}`,
      await obj.Body.transformToByteArray(),
    );
  }
  console.log("Files downloaded successfully.\n");
};
```

É hora de limpar seus recursos. Um bucket deve estar vazio para poder ser excluído. Essas duas funções esvaziam e excluem o bucket.

```
export const emptyBucket = async ({ bucketName }) => {
  const listObjectsCommand = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(listObjectsCommand);
  const keys = Contents.map((c) => c.Key);

  const deleteObjectsCommand = new DeleteObjectsCommand({
    Bucket: bucketName,
    Delete: { Objects: keys.map((key) => ({ Key: key })) },
  });
  await s3Client.send(deleteObjectsCommand);
  console.log(`${bucketName} emptied successfully.\n`);
};

export const deleteBucket = async ({ bucketName }) => {
  const command = new DeleteBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log(`${bucketName} deleted successfully.\n`);
};
```

A função “principal” reúne tudo. Se você executar esse arquivo diretamente, a função principal será chamada.

```
const main = async () => {
  const OBJECT_DIRECTORY = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../resources/sample_files/.sample_media`;

  try {
    console.log(wrapText("Welcome to the Amazon S3 getting started example."));
```

```
    console.log("Let's create a bucket.");
    const bucketName = await createBucket();
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("File upload."));
    console.log(
      "I have some default files ready to go. You can edit the source code to
 provide your own.",
    );
    await uploadFilesToBucket({
      bucketName,
      folderPath: OBJECT_DIRECTORY,
    });

    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Copy files."));
    await copyFileFromBucket({ destinationBucket: bucketName });
    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Download files."));
    await downloadFilesFromBucket({ bucketName });

    console.log(wrapText("Clean up."));
    await emptyBucket({ bucketName });
    await deleteBucket({ bucketName });
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject

- ListObjectsV2
- PutObject

Obter a configuração de retenção legal de um objeto

O exemplo de código a seguir mostra como obter a configuração de retenção legal de um bucket do S3.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

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

import { fileURLToPath } from "url";
import { GetObjectLegalHoldCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
  } catch (err) {
```

```
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "DOC-EXAMPLE-BUCKET", "OBJECT_KEY");
}
```

- Para obter detalhes da API, consulte GetObjectLegalHolda Referência AWS SDK for JavaScript da API.

Bloquear objetos do Amazon S3

O exemplo de código a seguir mostra como trabalhar com os recursos de bloqueio de objetos do S3.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

index.js- Ponto de entrada para o fluxo de trabalho. Isso orquestra todas as etapas. Visite GitHub para ver os detalhes de implementação do Cenário ScenarioInput ScenarioOutput,, ScenarioAction e.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "./welcome.steps.js";
import {
  confirmCreateBuckets,
```

```
  confirmPopulateBuckets,
  confirmSetLegalHoldFileEnabled,
  confirmSetLegalHoldFileRetention,
  confirmSetRetentionPeriodFileEnabled,
  confirmSetRetentionPeriodFileRetention,
  confirmUpdateLockPolicy,
  confirmUpdateRetention,
  createBuckets,
  createBucketsAction,
  populateBuckets,
  populateBucketsAction,
  setLegalHoldFileEnabledAction,
  setLegalHoldFileRetentionAction,
  setRetentionPeriodFileEnabledAction,
  setRetentionPeriodFileRetentionAction,
  updateLockPolicy,
  updateLockPolicyAction,
  updateRetention,
  updateRetentionAction,
} from "./setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
    deploy: new scenarios.Scenario(
      "S3 Object Locking - Deploy",
      [
        welcome(scenarios),
        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
        createBucketsAction(scenarios, client),
        updateRetention(scenarios),
        confirmUpdateRetention(scenarios),
        exitOnFalse(scenarios, "confirmUpdateRetention"),
        updateRetentionAction(scenarios, client),
        populateBuckets(scenarios),
```

```
        confirmPopulateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        updateLockPolicy(scenarios),
        confirmUpdateLockPolicy(scenarios),
        exitOnFalse(scenarios, "confirmUpdateLockPolicy"),
        updateLockPolicyAction(scenarios, client),
        confirmSetLegalHoldFileEnabled(scenarios),
        setLegalHoldFileEnabledAction(scenarios, client),
        confirmSetRetentionPeriodFileEnabled(scenarios),
        setRetentionPeriodFileEnabledAction(scenarios, client),
        confirmSetLegalHoldFileRetention(scenarios),
        setLegalHoldFileRetentionAction(scenarios, client),
        confirmSetRetentionPeriodFileRetention(scenarios),
        setRetentionPeriodFileRetentionAction(scenarios, client),
        saveState,
      ],
      initialState,
    ),
    demo: new scenarios.Scenario(
      "S3 Object Locking - Demo",
      [loadState, replAction(scenarios, client)],
      initialState,
    ),
    clean: new scenarios.Scenario(
      "S3 Object Locking - Destroy",
      [
        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
      ],
      initialState,
    ),
  };
};

// Call function if run directly
import { fileURLToPath } from "url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
```

```
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios);
}
```

welcome.steps.js- Envie mensagens de boas-vindas para o console.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    `Welcome to the Amazon Simple Storage Service (S3) Object Locking Workflow
 Scenario. For this workflow, we will use the AWS SDK for JavaScript to create
 several S3 buckets and files to demonstrate working with S3 locking features.`,
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };
```

setup.steps.js- Implante buckets, objetos e configurações de arquivos.

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

```
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const bucketPrefix = "js-object-locking";

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    `The following buckets will be created:
         ${bucketPrefix}-no-lock with object lock False.
         ${bucketPrefix}-lock-enabled with object lock True.
         ${bucketPrefix}-retention-after-creation with object lock False.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });
```

```
/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const noLockBucketName = `${bucketPrefix}-no-lock`;
    const lockEnabledBucketName = `${bucketPrefix}-lock-enabled`;
    const retentionBucketName = `${bucketPrefix}-retention-after-creation`;

    await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
    await client.send(
      new CreateBucketCommand({
        Bucket: lockEnabledBucketName,
        ObjectLockEnabledForBucket: true,
      }),
    );
    await client.send(new CreateBucketCommand({ Bucket: retentionBucketName }));

    state.noLockBucketName = noLockBucketName;
    state.lockEnabledBucketName = lockEnabledBucketName;
    state.retentionBucketName = retentionBucketName;
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    `The following test files will be created:
         file0.txt in ${bucketPrefix}-no-lock.
         file1.txt in ${bucketPrefix}-no-lock.
         file0.txt in ${bucketPrefix}-lock-enabled.
         file1.txt in ${bucketPrefix}-lock-enabled.
         file0.txt in ${bucketPrefix}-retention-after-creation.
         file1.txt in ${bucketPrefix}-retention-after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
```

```
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
```

```
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateRetention = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateRetention",
    `A bucket can be configured to use object locking with a default retention
 period.
   A default retention period will be configured for ${bucketPrefix}-retention-
after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateRetention",
    "Configure default retention period?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
```

```
 * @param {S3Client} client
 */
const updateRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateRetentionAction", async (state) => {
    await client.send(
      new PutBucketVersioningCommand({
        Bucket: state.retentionBucketName,
        VersioningConfiguration: {
          MFADelete: MFADeleteStatus.Disabled,
          Status: BucketVersioningStatus.Enabled,
        },
      }),
    );

    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.retentionBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
          Rule: {
            DefaultRetention: {
              Mode: "GOVERNANCE",
              Years: 1,
            },
          },
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateLockPolicy = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateLockPolicy",
    `Object lock policies can also be added to existing buckets.
   An object lock policy will be added to ${bucketPrefix}-lock-enabled.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
```

```
const confirmUpdateLockPolicy = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateLockPolicy",
    "Add object lock policy?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateLockPolicyAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateLockPolicyAction", async (state) => {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.lockEnabledBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileEnabled",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.lockEnabledBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
```

```
    "setLegalHoldFileEnabledAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.lockEnabledBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetRetentionPeriodFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileEnabled",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.lockEnabledBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileEnabledAction",
    async (state) => {
      const retentionDate = new Date();
```

```
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
        }),
      );
      console.log(
        `Set retention for file1.txt in ${state.lockEnabledBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.retentionBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileRetentionAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
```

```
          Bucket: state.retentionBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.retentionBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileRetention },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmSetRetentionPeriodFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileRetention",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.retentionBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileRetentionAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.retentionBucketName,
          Key: "file1.txt",
```

```
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
          BypassGovernanceRetention: true,
        }),
      );
      console.log(
        `Set retention for file1.txt in ${state.retentionBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileRetention },
  );

export {
  createBuckets,
  confirmCreateBuckets,
  createBucketsAction,
  populateBuckets,
  confirmPopulateBuckets,
  populateBucketsAction,
  updateRetention,
  confirmUpdateRetention,
  updateRetentionAction,
  updateLockPolicy,
  confirmUpdateLockPolicy,
  updateLockPolicyAction,
  confirmSetLegalHoldFileEnabled,
  setLegalHoldFileEnabledAction,
  confirmSetRetentionPeriodFileEnabled,
  setRetentionPeriodFileEnabledAction,
  confirmSetLegalHoldFileRetention,
  setLegalHoldFileRetentionAction,
  confirmSetRetentionPeriodFileRetention,
  setRetentionPeriodFileRetentionAction,
};
```

repl.steps.js- Visualize e exclua arquivos nos buckets.

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

```
import {
  ChecksumAlgorithm,
  DeleteObjectCommand,
  GetObjectLegalHoldCommand,
  GetObjectLockConfigurationCommand,
  GetObjectRetentionCommand,
  ListObjectVersionsCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  DELETE_FILE: 2,
  DELETE_FILE_WITH_RETENTION: 3,
  OVERWRITE_FILE: 4,
  VIEW_RETENTION_SETTINGS: 5,
  VIEW_LEGAL_HOLD_SETTINGS: 6,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new scenarios.ScenarioInput(
    "replChoice",
    `Explore the S3 locking features by selecting one of the following choices`,
    {
      type: "select",
      choices: [
        { name: "List all files in buckets", value: choices.LIST_ALL_FILES },
        { name: "Attempt to delete a file.", value: choices.DELETE_FILE },
        {
          name: "Attempt to delete a file with retention period bypass.",
          value: choices.DELETE_FILE_WITH_RETENTION,
        },
```

```
        { name: "Attempt to overwrite a file.", value: choices.OVERWRITE_FILE },
        {
          name: "View the object and bucket retention settings for a file.",
          value: choices.VIEW_RETENTION_SETTINGS,
        },
        {
          name: "View the legal hold settings for a file.",
          value: choices.VIEW_LEGAL_HOLD_SETTINGS,
        },
        { name: "Finish the workflow.", value: choices.EXIT },
      ],
    },
  );

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;
      files.push({ bucket, key: Key, version: VersionId });
    }
  }

  return files;
};

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const replAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.noLockBucketName,
```

```
      state.lockEnabledBucketName,
      state.retentionBucketName,
    ]);

    const fileInput = new scenarios.ScenarioInput(
      "selectedFile",
      "Select a file:",
      {
        type: "select",
        choices: files.map((file, index) => ({
          name: `${index + 1}: ${file.bucket}: ${file.key} (version: ${
            file.version
          })`,
          value: index,
        })),
      },
    );

    const { replChoice } = state;

    switch (replChoice) {
      case choices.LIST_ALL_FILES: {
        const files = await getAllFiles(client, [
          state.noLockBucketName,
          state.lockEnabledBucketName,
          state.retentionBucketName,
        ]);
        state.replOutput = files
          .map(
            (file) =>
              `${file.bucket}: ${file.key} (version: ${file.version})`,
          )
          .join("\n");
        break;
      }
      case choices.DELETE_FILE: {
        /** @type {number} */
        const fileToDelete = await fileInput.handle(state);
        const selectedFile = files[fileToDelete];
        try {
          await client.send(
            new DeleteObjectCommand({
              Bucket: selectedFile.bucket,
              Key: selectedFile.key,
```

```
              VersionId: selectedFile.version,
            }),
          );
          state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
        } catch (err) {
          state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
        }
        break;
      }
      case choices.DELETE_FILE_WITH_RETENTION: {
        /** @type {number} */
        const fileToDelete = await fileInput.handle(state);
        const selectedFile = files[fileToDelete];
        try {
          await client.send(
            new DeleteObjectCommand({
              Bucket: selectedFile.bucket,
              Key: selectedFile.key,
              VersionId: selectedFile.version,
              BypassGovernanceRetention: true,
            }),
          );
          state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
        } catch (err) {
          state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
        }
        break;
      }
      case choices.OVERWRITE_FILE: {
        /** @type {number} */
        const fileToOverwrite = await fileInput.handle(state);
        const selectedFile = files[fileToOverwrite];
        try {
          await client.send(
            new PutObjectCommand({
              Bucket: selectedFile.bucket,
              Key: selectedFile.key,
              Body: "New content",
              ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
            }),
```

```
            );
            state.replOutput = `Overwrote ${selectedFile.key} in
 ${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to overwrite object ${selectedFile.key} in
 bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.VIEW_RETENTION_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
            const retention = await client.send(
              new GetObjectRetentionCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            const bucketConfig = await client.send(
              new GetObjectLockConfigurationCommand({
                Bucket: selectedFile.bucket,
              }),
            );
            state.replOutput = `Object retention for ${selectedFile.key}
 in ${selectedFile.bucket}: ${retention.Retention?.Mode} until
 ${retention.Retention?.RetainUntilDate?.toISOString()}.
Bucket object lock config for ${selectedFile.bucket} in ${selectedFile.bucket}:
Enabled: ${bucketConfig.ObjectLockConfiguration?.ObjectLockEnabled}
Rule:
 ${JSON.stringify(bucketConfig.ObjectLockConfiguration?.Rule?.DefaultRetention)}`;
          } catch (err) {
            state.replOutput = `Unable to fetch object lock retention:
 '${err.message}'`;
          }
          break;
        }
        case choices.VIEW_LEGAL_HOLD_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
```

```
            const legalHold = await client.send(
              new GetObjectLegalHoldCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Object legal hold for ${selectedFile.key} in
 ${selectedFile.bucket}: Status: ${legalHold.LegalHold?.Status}`;
          } catch (err) {
            state.replOutput = `Unable to fetch legal hold: '${err.message}'`;
          }
          break;
        }
        default:
          throw new Error(`Invalid replChoice: ${replChoice}`);
      }
    },
    {
      whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new scenarios.ScenarioOutput(
          "REPL output",
          (state) => state.replOutput,
          { preformatted: true },
        ),
      },
    },
  );

export { replInput, replAction, choices };
```

clean.steps.js- Destrua todos os recursos criados.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  DeleteObjectCommand,
  DeleteBucketCommand,
  ListObjectVersionsCommand,
  GetObjectLegalHoldCommand,
```

```
  GetObjectRetentionCommand,
  PutObjectLegalHoldCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { noLockBucketName, lockEnabledBucketName, retentionBucketName } =
      state;

    const buckets = [
      noLockBucketName,
      lockEnabledBucketName,
      retentionBucketName,
    ];

    for (const bucket of buckets) {
      /** @type {import("@aws-sdk/client-s3").ListObjectVersionsCommandOutput} */
      let objectsResponse;

      try {
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
```

```
      );
    } catch (e) {
      if (e instanceof Error && e.name === "NoSuchBucket") {
        console.log("Object's bucket has already been deleted.");
        continue;
      } else {
        throw e;
      }
    }

    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;

      try {
        const legalHold = await client.send(
          new GetObjectLegalHoldCommand({
            Bucket: bucket,
            Key,
            VersionId,
          }),
        );

        if (legalHold.LegalHold?.Status === "ON") {
          await client.send(
            new PutObjectLegalHoldCommand({
              Bucket: bucket,
              Key,
              VersionId,
              LegalHold: {
                Status: "OFF",
              },
            }),
          );
        }
      } catch (err) {
        console.log(
          `Unable to fetch legal hold for ${Key} in ${bucket}: '${err.message}'`,
        );
      }

      try {
        const retention = await client.send(
          new GetObjectRetentionCommand({
            Bucket: bucket,
```

```
              Key,
              VersionId,
            }),
          );

          if (retention.Retention?.Mode === "GOVERNANCE") {
            await client.send(
              new DeleteObjectCommand({
                Bucket: bucket,
                Key,
                VersionId,
                BypassGovernanceRetention: true,
              }),
            );
          }
        } catch (err) {
          console.log(
            `Unable to fetch object lock retention for ${Key} in ${bucket}:
 '${err.message}'`,
          );
        }

        await client.send(
          new DeleteObjectCommand({
            Bucket: bucket,
            Key,
            VersionId,
          }),
        );
      }

      await client.send(new DeleteBucketCommand({ Bucket: bucket }));
      console.log(`Delete for ${bucket} complete.`);
    }
  });

export { confirmCleanup, cleanupAction };
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - GetObjectLegalHold

- GetObjectLockConfiguration
- GetObjectRetenção
- PutObjectLegalHold
- PutObjectLockConfiguration
- PutObjectRetenção

Fazer upload ou download de arquivos grandes

O exemplo de código a seguir mostra como fazer upload ou download de arquivos grandes de e para o Amazon S3.

Para obter mais informações, consulte Carregar um objeto usando carregamento fracionado.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Faça upload de um arquivo grande.

```
import {
  CreateMultipartUploadCommand,
  UploadPartCommand,
  CompleteMultipartUploadCommand,
  AbortMultipartUploadCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

export const main = async () => {
  const s3Client = new S3Client({});
  const bucketName = "test-bucket";
```

```
const key = "multipart.txt";
const str = createString();
const buffer = Buffer.from(str, "utf8");

let uploadId;

try {
  const multipartUpload = await s3Client.send(
    new CreateMultipartUploadCommand({
      Bucket: bucketName,
      Key: key,
    }),
  );

  uploadId = multipartUpload.UploadId;

  const uploadPromises = [];
  // Multipart uploads require a minimum size of 5 MB per part.
  const partSize = Math.ceil(buffer.length / 5);

  // Upload each part.
  for (let i = 0; i < 5; i++) {
    const start = i * partSize;
    const end = start + partSize;
    uploadPromises.push(
      s3Client
        .send(
          new UploadPartCommand({
            Bucket: bucketName,
            Key: key,
            UploadId: uploadId,
            Body: buffer.subarray(start, end),
            PartNumber: i + 1,
          }),
        )
        .then((d) => {
          console.log("Part", i + 1, "uploaded");
          return d;
        }),
    );
  }

  const uploadResults = await Promise.all(uploadPromises);
```

```
    return await s3Client.send(
      new CompleteMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
        UploadId: uploadId,
        MultipartUpload: {
          Parts: uploadResults.map(({ ETag }, i) => ({
            ETag,
            PartNumber: i + 1,
          })),
        },
      }),
    );

    // Verify the output by downloading the file from the Amazon Simple Storage
 Service (Amazon S3) console.
    // Because the output is a 25 MB string, text editors might struggle to open the
 file.
  } catch (err) {
    console.error(err);

    if (uploadId) {
      const abortCommand = new AbortMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
        UploadId: uploadId,
      });

      await s3Client.send(abortCommand);
    }
  }
};
```

Baixe um arquivo grande.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream } from "fs";

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;

export const getObjectRange = ({ bucket, key, start, end }) => {
```

```
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

/**
 * @param {string | undefined} contentRange
 */
export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: parseInt(start),
    end: parseInt(end),
    length: parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

// When downloading a large file, you might want to break it down into
// smaller pieces. Amazon S3 accepts a Range header to specify the start
// and end of the byte range to be downloaded.
const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url)),
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    console.log(`Downloading bytes ${nextRange.start} to ${nextRange.end}`);

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
```

```
    });

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

export const main = async () => {
  await downloadInChunks({
    bucket: "my-cool-bucket",
    key: "my-cool-object.txt",
  });
};
```

Exemplos sem servidor

Invocar uma função do Lambda em um acionador do Amazon S3

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo upload de um objeto para 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 o tipo de conteúdo do objeto.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do S3 com o uso do JavaScript Lambda.

```
// 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);
    }
};
```

Consumindo um evento do S3 com o uso do TypeScript Lambda.

```
// 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);
  }
};
```

Exemplos do S3 Glacier usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o S3 Glacier.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`CreateVault`

O código de exemplo a seguir mostra como usar `CreateVault`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Crie o cofre.

```
// Load the SDK for JavaScript
import { CreateVaultCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "./libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME
const params = { vaultName: vaultname };

const run = async () => {
  try {
    const data = await glacierClient.send(new CreateVaultCommand(params));
    console.log("Success, vault created!");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error");
  }
};
run();
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.

- Para obter detalhes da API, consulte CreateVaulta Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
// Call Glacier to create the vault
glacier.createVault({ vaultName: "YOUR_VAULT_NAME" }, function (err) {
  if (!err) {
    console.log("Created vault!");
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateVaulta Referência AWS SDK for JavaScript da API.

`UploadArchive`

O código de exemplo a seguir mostra como usar `UploadArchive`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Faça upload do arquivo.

```
// Load the SDK for JavaScript
import { UploadArchiveCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "./libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME

// Create a new service object and buffer
const buffer = new Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer
const params = { vaultName: vaultname, body: buffer };

const run = async () => {
  try {
    const data = await glacierClient.send(new UploadArchiveCommand(params));
    console.log("Archive ID", data.archiveId);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error uploading archive!", err);
  }
};
run();
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte UploadArchivea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object and buffer
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
buffer = Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer

var params = { vaultName: "YOUR_VAULT_NAME", body: buffer };
// Call Glacier to upload the archive.
glacier.uploadArchive(params, function (err, data) {
  if (err) {
    console.log("Error uploading archive!", err);
  } else {
    console.log("Archive ID", data.archiveId);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte UploadArchivea Referência AWS SDK for JavaScript da API.

SageMaker exemplos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com SageMaker.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá SageMaker

Os exemplos de código a seguir mostram como começar a usar SageMaker.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import {
  SageMakerClient,
  ListNotebookInstancesCommand,
} from "@aws-sdk/client-sagemaker";

const client = new SageMakerClient({
  region: "us-west-2",
});

export const helloSagemaker = async () => {
  const command = new ListNotebookInstancesCommand({ MaxResults: 5 });

  const response = await client.send(command);
  console.log(
    "Hello Amazon SageMaker! Let's list some of your notebook instances:",
  );

  const instances = response.NotebookInstances || [];

  if (instances.length === 0) {
    console.log(
      "• No notebook instances found. Try creating one in the AWS Management Console
 or with the CreateNotebookInstanceCommand.",
```

```
    );
  } else {
    console.log(
      instances
        .map(
          (i) =>
            `• Instance: ${i.NotebookInstanceName}\n  Arn:${
              i.NotebookInstanceArn
            } \n  Creation Date: ${i.CreationTime.toISOString()}`,
        )
        .join("\n"),
    );
  }

  return response;
};
```

- Para obter detalhes da API, consulte ListNotebookInstâncias na Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários

Ações

`CreatePipeline`

O código de exemplo a seguir mostra como usar `CreatePipeline`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Uma função que cria um SageMaker pipeline usando uma definição JSON fornecida localmente.

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
```

```
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}
```

- Para obter detalhes da API, consulte CreatePipelinea Referência AWS SDK for JavaScript da API.

`DeletePipeline`

O código de exemplo a seguir mostra como usar `DeletePipeline`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

A sintaxe para excluir um SageMaker pipeline. Esse código faz parte de uma função maior. Consulte “Criar um pipeline” ou o GitHub repositório para obter mais contexto.

```
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
```

- Para obter detalhes da API, consulte DeletePipelinea Referência AWS SDK for JavaScript da API.

`DescribePipelineExecution`

O código de exemplo a seguir mostra como usar `DescribePipelineExecution`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Aguarde até que a execução de um SageMaker pipeline seja bem-sucedida, falhe ou pare.

```
/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
```

```
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- Para obter detalhes da API, consulte DescribePipelineExecução na Referência AWS SDK for JavaScript da API.

`StartPipelineExecution`

O código de exemplo a seguir mostra como usar `StartPipelineExecution`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Inicie a execução de um SageMaker pipeline.

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
```

```
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}
```

- Para obter detalhes da API, consulte StartPipelineExecução na Referência AWS SDK for JavaScript da API.

Cenários

Conceitos básicos de trabalhos geoespaciais e pipelines

O exemplo de código a seguir mostra como:

- Configurar recursos para um pipeline.
- Configurar um pipeline que executa um trabalho geoespacial.
- Iniciar a execução de um pipeline.
- Monitorar o status da execução.
- Ver a saída do pipeline.
- Limpar recursos.

Para obter mais informações, consulte Criar e executar SageMaker pipelines usando AWS SDKs no Community.aws.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

O trecho do arquivo a seguir contém funções que usam o SageMaker cliente para gerenciar um pipeline.

```
import { readFileSync } from "fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
```

```
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
  GetRoleCommand,
  ListPoliciesCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
  GetFunctionCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-
geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
  GetQueueUrlCommand,
```

```
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 props
 */
export async function createLambdaExecutionRole({ name, iamClient }) {
  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: { Service: ["lambda.amazonaws.com"] },
            },
          ],
        }),
      }),
    );

  let role = null;

  try {
    const { Role } = await createRole();
    role = Role;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
```

```
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 Amazon SageMaker.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: [
          "sqs:ReceiveMessage",
          "sqs:DeleteMessage",
          "sqs:GetQueueAttributes",
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents",
          "sagemaker-geospatial:StartVectorEnrichmentJob",
          "sagemaker-geospatial:GetVectorEnrichmentJob",
          "sagemaker:SendPipelineExecutionStepFailure",
          "sagemaker:SendPipelineExecutionStepSuccess",
          "sagemaker-geospatial:ExportVectorEnrichmentJob",
        ],
        Resource: "*",
      },
```

```
      {
        Effect: "Allow",
        // The AWS Lambda function needs permission to pass the pipeline execution
 role to
        // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
 function
        // from elevating privileges. For more information, see:
        // https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_use_passrole.html
        Action: ["iam:PassRole"],
        Resource: `${pipelineExecutionRoleArn}`,
        Condition: {
          StringEquals: {
            "iam:PassedToService": [
              "sagemaker.amazonaws.com",
              "sagemaker-geospatial.amazonaws.com",
            ],
          },
        },
      },
    ],
  };

  const createPolicy = () =>
    iamClient.send(
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
        PolicyName: name,
      }),
    );

  let policy = null;

  try {
    const { Policy } = await createPolicy();
    policy = Policy;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
        policy = Policies.find((p) => p.PolicyName === name);
```

```
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}

/**
 * Attach an AWS IAM policy to an AWS IAM role.
 * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-
iam').IAMClient}} props
 */
export async function attachPolicy({ roleName, policyArn, iamClient }) {
  const attachPolicyCommand = new AttachRolePolicyCommand({
    RoleName: roleName,
    PolicyArn: policyArn,
  });

  await iamClient.send(attachPolicyCommand);
  return {
    cleanUp: async () => {
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: roleName,
          PolicyArn: policyArn,
        }),
      );
    },
  };
}

/**
 * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon
 SageMaker Geospatial clients
```

```
 * in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
 Amazon SageMaker
 * Geospatial client wasn't introduced until v3.221.0.
 * @param {{ name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient }} props
 */
export async function createLambdaLayer({ name, lambdaClient }) {
  const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;
  const { LayerVersionArn, Version } = await lambdaClient.send(
    new PublishLayerVersionCommand({
      LayerName: name,
      Content: {
        ZipFile: Uint8Array.from(readFileSync(layerPath)),
      },
    }),
  );

  return {
    versionArn: LayerVersionArn,
    version: Version,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteLayerVersionCommand({
          LayerName: name,
          VersionNumber: Version,
        }),
      );
    },
  };
}

/**
 * Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
 pipeline
 * execution steps.
 * @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient, layerVersionArn: string}} props
 */
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
  layerVersionArn,
}) {
```

```
const lambdaPath = `${dirnameFromMetaUrl(
  import.meta.url,
)}lambda/dist/index.mjs.zip`;

// If a function of the same name already exists, return that
// function's ARN instead. By default this is
// "sagemaker-wkflw-lambda-function", so collisions are
// unlikely.
const createFunction = async () => {
  try {
    return await lambdaClient.send(
      new CreateFunctionCommand({
        Code: {
          ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
        },
        Runtime: Runtime.nodejs18x,
        Handler: "index.handler",
        Layers: [layerVersionArn],
        FunctionName: name,
        Role: roleArn,
      }),
    );
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceConflictException"
    ) {
      const { Configuration } = await lambdaClient.send(
        new GetFunctionCommand({ FunctionName: name }),
      );
      return Configuration;
    } else {
      throw caught;
    }
  }
};

// Function creation fails if the Role is not ready. This retries
// function creation until it succeeds or it times out.
const { FunctionArn } = await retry(
  { intervalInMs: 1000, maxRetries: 60 },
  createFunction,
);
```

```
  return {
    arn: FunctionArn,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteFunctionCommand({ FunctionName: name }),
      );
    },
  };
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
 props
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
  const s3Path = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../../workflows/sagemaker_pipelines/resources/latlongtest.csv`;

  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "input/sample_data.csv",
      Body: readFileSync(s3Path),
    }),
  );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient, wait:
 (ms: number) => Promise<void>}} props
 */
export async function createSagemakerRole({ name, iamClient, wait }) {
  let role = null;

  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
```

```
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: {
                Service: [
                  "sagemaker.amazonaws.com",
                  "sagemaker-geospatial.amazonaws.com",
                ],
              },
            },
          ],
        }),
      }),
    );

  try {
    const { Role } = await createRole();
    role = Role;
    // Wait for the role to be ready.
    await wait(10);
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}
```

```
/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 messages to
 * the Amazon SQS queue.
 * @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
 import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
 */
export async function createSagemakerExecutionPolicy({
  sqsQueueArn,
  lambdaArn,
  iamClient,
  name,
  s3BucketName,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: ["lambda:InvokeFunction"],
        Resource: lambdaArn,
      },
      {
        Effect: "Allow",
        Action: ["s3:*"],
        Resource: [
          `arn:aws:s3:::${s3BucketName}`,
          `arn:aws:s3:::${s3BucketName}/*`,
        ],
      },
      {
        Effect: "Allow",
        Action: ["sqs:SendMessage"],
        Resource: sqsQueueArn,
      },
    ],
  };

  const createPolicy = () =>
    iamClient.send(
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
```

```
        PolicyName: name,
      }),
    );

  let policy = null;

  try {
    const { Policy } = await createPolicy();
    policy = Policy;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}

/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
```

```
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
```

```
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
  const createSqsQueue = () =>
    sqsClient.send(
      new CreateQueueCommand({
        QueueName: name,
        Attributes: {
          DelaySeconds: "5",
          ReceiveMessageWaitTimeSeconds: "5",
          VisibilityTimeout: "300",
        },
      }),
    );

  let queueUrl = null;
  try {
    const { QueueUrl } = await createSqsQueue();
    queueUrl = QueueUrl;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "QueueNameExists") {
      const { QueueUrl } = await sqsClient.send(
        new GetQueueUrlCommand({ QueueName: name }),
      );
      queueUrl = QueueUrl;
    } else {
      throw caught;
```

```
    }
  }

  const { Attributes } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    () =>
      sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: queueUrl,
          AttributeNames: ["QueueArn"],
        }),
      ),
  );

  return {
    queueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl: queueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{
 *   paginateListEventSourceMappings: () => Generator<import('@aws-sdk/client-
lambda').ListEventSourceMappingsCommandOutput>,
 *   lambdaName: string,
 *   queueArn: string,
 *   lambdaClient: import('@aws-sdk/client-lambda').LambdaClient}} props
 */
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
  paginateListEventSourceMappings,
}) {
  let uuid = null;
  const createEvenSourceMapping = () =>
    lambdaClient.send(
      new CreateEventSourceMappingCommand({
        EventSourceArn: queueArn,
```

```
      FunctionName: lambdaName,
    }),
  );

try {
  const { UUID } = await createEvenSourceMapping();
  uuid = UUID;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "ResourceConflictException"
  ) {
    const paginator = paginateListEventSourceMappings(
      { client: lambdaClient },
      {},
    );
    /**
     * @type {import('@aws-sdk/client-lambda').EventSourceMappingConfiguration[]}
     */
    const eventSourceMappings = [];
    for await (const page of paginator) {
      eventSourceMappings.concat(page.EventSourceMappings || []);
    }

    const { Configuration } = await lambdaClient.send(
      new GetFunctionCommand({ FunctionName: lambdaName }),
    );

    uuid = eventSourceMappings.find(
      (mapping) =>
        mapping.EventSourceArn === queueArn &&
        mapping.FunctionArn === Configuration.FunctionArn,
    ).UUID;
  } else {
    throw caught;
  }
}

return {
  cleanUp: async () => {
    await lambdaClient.send(
      new DeleteEventSourceMappingCommand({
        UUID: uuid,
      }),
```

```
      );
    },
  };
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{
 *   s3Client: import('@aws-sdk/client-s3').S3Client,
 *   name: string,
 *   paginateListObjectsV2: () => Generator<import('@aws-sdk/client-
s3').ListObjectsCommandOutput>
 * }} props
 */
export async function createS3Bucket({
  name,
  s3Client,
  paginateListObjectsV2,
}) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(
        { client: s3Client },
        { Bucket: name },
      );
      for await (const page of paginator) {
        const objects = page.Contents;
        if (objects) {
          for (const object of objects) {
            await s3Client.send(
              new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
            );
          }
        }
      }
      await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
    },
  };
}

/**
```

```
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };
```

```
  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}

/**
```

```
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-
s3').S3Client}} param0
```

```
 */
export async function getObject({ bucket, s3Client }) {
  const prefix = "output/";
  const { Contents } = await s3Client.send(
    new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
  );

  if (!Contents.length) {
    throw new Error("No objects found in bucket.");
  }

  // Find the CSV file.
  const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

  if (!outputObject) {
    throw new Error(`No CSV file found in bucket with the prefix "${prefix}".`);
  }

  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: outputObject.Key,
    }),
  );

  return Body.transformToString();
}
```

Essa função é um trecho de um arquivo que usa as funções anteriores da biblioteca para configurar um SageMaker pipeline, executá-lo e excluir todos os recursos criados.

```
import { retry, wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  attachPolicy,
  configureLambdaSQSEventSource,
  createLambdaExecutionPolicy,
  createLambdaExecutionRole,
  createLambdaFunction,
  createLambdaLayer,
  createS3Bucket,
  createSQSQueue,
  createSagemakerExecutionPolicy,
```

```
  createSagemakerPipeline,
  createSagemakerRole,
  getObject,
  startPipelineExecution,
  uploadCSVDataToS3,
  waitForPipelineComplete,
} from "./lib.js";
import { MESSAGES } from "./messages.js";

export class SageMakerPipelinesWkflw {
  names = {
    LAMBDA_EXECUTION_ROLE: "sagemaker-wkflw-lambda-execution-role",
    LAMBDA_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-lambda-execution-role-policy",
    LAMBDA_FUNCTION: "sagemaker-wkflw-lambda-function",
    LAMBDA_LAYER: "sagemaker-wkflw-lambda-layer",
    SAGE_MAKER_EXECUTION_ROLE: "sagemaker-wkflw-pipeline-execution-role",
    SAGE_MAKER_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-pipeline-execution-role-policy",
    SAGE_MAKER_PIPELINE: "sagemaker-wkflw-pipeline",
    SQS_QUEUE: "sagemaker-wkflw-sqs-queue",
    S3_BUCKET: `sagemaker-wkflw-s3-bucket-${Date.now()}`,
  };

  cleanUpFunctions = [];

  /**
   * @param {import("@aws-doc-sdk-examples/lib/prompter.js").Prompter} prompter
   * @param {import("@aws-doc-sdk-examples/lib/logger.js").Logger} logger
   * @param {{ IAM: import("@aws-sdk/client-iam").IAMClient, Lambda: import("@aws-
sdk/client-lambda").LambdaClient, SageMaker: import("@aws-sdk/client-
sagemaker").SageMakerClient, S3: import("@aws-sdk/client-s3").S3Client, SQS:
 import("@aws-sdk/client-sqs").SQSClient }} clients
   */
  constructor(prompter, logger, clients) {
    this.prompter = prompter;
    this.logger = logger;
    this.clients = clients;
  }

  async run() {
    try {
      await this.startWorkflow();
    } catch (err) {
```

```
      console.error(err);
      throw err;
    } finally {
      this.logger.logSeparator();
      const doCleanUp = await this.prompter.confirm({
        message: "Clean up resources?",
      });
      if (doCleanUp) {
        await this.cleanUp();
      }
    }
  }

  async cleanUp() {
    // Run all of the clean up functions. If any fail, we log the error and
continue.
    // This ensures all clean up functions are run.
    for (let i = this.cleanUpFunctions.length - 1; i >= 0; i--) {
      await retry(
        { intervalInMs: 1000, maxRetries: 60, swallowError: true },
        this.cleanUpFunctions[i],
      );
    }
  }

  async startWorkflow() {
    this.logger.logSeparator(MESSAGES.greetingHeader);
    await this.logger.log(MESSAGES.greeting);

    this.logger.logSeparator();
    await this.logger.log(
      MESSAGES.creatingRole.replace(
        "${ROLE_NAME}",
        this.names.LAMBDA_EXECUTION_ROLE,
      ),
    );

    // Create an IAM role that will be assumed by the AWS Lambda function. This
function
    // is triggered by Amazon SQS messages and calls SageMaker and SageMaker
GeoSpatial actions.
    const { arn: lambdaExecutionRoleArn, cleanUp: lambdaExecutionRoleCleanUp } =
      await createLambdaExecutionRole({
        name: this.names.LAMBDA_EXECUTION_ROLE,
```

```
        iamClient: this.clients.IAM,
      });
    // Add a clean up step to a stack for every resource created.
    this.cleanUpFunctions.push(lambdaExecutionRoleCleanUp);

    await this.logger.log(
      MESSAGES.roleCreated.replace(
        "${ROLE_NAME}",
        this.names.LAMBDA_EXECUTION_ROLE,
      ),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingRole.replace(
        "${ROLE_NAME}",
        this.names.SAGE_MAKER_EXECUTION_ROLE,
      ),
    );

    // Create an IAM role that will be assumed by the SageMaker pipeline. The
pipeline
    // sends messages to an Amazon SQS queue and puts/retrieves Amazon S3 objects.
    const {
      arn: pipelineExecutionRoleArn,
      cleanUp: pipelineExecutionRoleCleanUp,
    } = await createSagemakerRole({
      iamClient: this.clients.IAM,
      name: this.names.SAGE_MAKER_EXECUTION_ROLE,
      wait,
    });
    this.cleanUpFunctions.push(pipelineExecutionRoleCleanUp);

    await this.logger.log(
      MESSAGES.roleCreated.replace(
        "${ROLE_NAME}",
        this.names.SAGE_MAKER_EXECUTION_ROLE,
      ),
    );

    this.logger.logSeparator();
```

```
    // Create an IAM policy that allows the AWS Lambda function to invoke SageMaker
APIs.
    const {
      arn: lambdaExecutionPolicyArn,
      policy: lambdaPolicy,
      cleanUp: lambdaExecutionPolicyCleanUp,
    } = await createLambdaExecutionPolicy({
      name: this.names.LAMBDA_EXECUTION_ROLE_POLICY,
      s3BucketName: this.names.S3_BUCKET,
      iamClient: this.clients.IAM,
      pipelineExecutionRoleArn,
    });
    this.cleanUpFunctions.push(lambdaExecutionPolicyCleanUp);

    console.log(JSON.stringify(lambdaPolicy, null, 2), "\n");

    await this.logger.log(
      MESSAGES.attachPolicy
        .replace("${POLICY_NAME}", this.names.LAMBDA_EXECUTION_ROLE_POLICY)
        .replace("${ROLE_NAME}", this.names.LAMBDA_EXECUTION_ROLE),
    );

    await this.prompter.checkContinue();

    // Attach the Lambda execution policy to the execution role.
    const { cleanUp: lambdaExecutionRolePolicyCleanUp } = await attachPolicy({
      roleName: this.names.LAMBDA_EXECUTION_ROLE,
      policyArn: lambdaExecutionPolicyArn,
      iamClient: this.clients.IAM,
    });
    this.cleanUpFunctions.push(lambdaExecutionRolePolicyCleanUp);

    await this.logger.log(MESSAGES.policyAttached);

    this.logger.logSeparator();

    // Create Lambda layer for SageMaker packages.
    const { versionArn: layerVersionArn, cleanUp: lambdaLayerCleanUp } =
      await createLambdaLayer({
        name: this.names.LAMBDA_LAYER,
        lambdaClient: this.clients.Lambda,
      });
    this.cleanUpFunctions.push(lambdaLayerCleanUp);
```

```
    await this.logger.log(
      MESSAGES.creatingFunction.replace(
        "${FUNCTION_NAME}",
        this.names.LAMBDA_FUNCTION,
      ),
    );

    // Create the Lambda function with the execution role.
    const { arn: lambdaArn, cleanUp: lambdaCleanUp } =
      await createLambdaFunction({
        roleArn: lambdaExecutionRoleArn,
        lambdaClient: this.clients.Lambda,
        name: this.names.LAMBDA_FUNCTION,
        layerVersionArn,
      });
    this.cleanUpFunctions.push(lambdaCleanUp);

    await this.logger.log(
      MESSAGES.functionCreated.replace(
        "${FUNCTION_NAME}",
        this.names.LAMBDA_FUNCTION,
      ),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingSQSQueue.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
    );

    // Create an SQS queue for the SageMaker pipeline.
    const {
      queueUrl,
      queueArn,
      cleanUp: queueCleanUp,
    } = await createSQSQueue({
      name: this.names.SQS_QUEUE,
      sqsClient: this.clients.SQS,
    });
    this.cleanUpFunctions.push(queueCleanUp);

    await this.logger.log(
      MESSAGES.sqsQueueCreated.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
    );
```

```
this.logger.logSeparator();

await this.logger.log(
  MESSAGES.configuringLambdaSQSEventSource
    .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
    .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

// Configure the SQS queue as an event source for the Lambda.
const { cleanUp: lambdaSQSEventSourceCleanUp } =
  await configureLambdaSQSEventSource({
    lambdaArn,
    lambdaName: this.names.LAMBDA_FUNCTION,
    queueArn,
    sqsClient: this.clients.SQS,
    lambdaClient: this.clients.Lambda,
  });
this.cleanUpFunctions.push(lambdaSQSEventSourceCleanUp);

await this.logger.log(
  MESSAGES.lambdaSQSEventSourceConfigured
    .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
    .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

this.logger.logSeparator();

// Create an IAM policy that allows the SageMaker pipeline to invoke AWS Lambda
// and send messages to the Amazon SQS queue.
const {
  arn: pipelineExecutionPolicyArn,
  policy: sagemakerPolicy,
  cleanUp: pipelineExecutionPolicyCleanUp,
} = await createSagemakerExecutionPolicy({
  sqsQueueArn: queueArn,
  lambdaArn,
  iamClient: this.clients.IAM,
  name: this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY,
  s3BucketName: this.names.S3_BUCKET,
});
this.cleanUpFunctions.push(pipelineExecutionPolicyCleanUp);

console.log(JSON.stringify(sagemakerPolicy, null, 2));
```

```
    await this.logger.log(
      MESSAGES.attachPolicy
        .replace("${POLICY_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY)
        .replace("${ROLE_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE),
    );

    await this.prompter.checkContinue();

    // Attach the SageMaker execution policy to the execution role.
    const { cleanUp: pipelineExecutionRolePolicyCleanUp } = await attachPolicy({
      roleName: this.names.SAGE_MAKER_EXECUTION_ROLE,
      policyArn: pipelineExecutionPolicyArn,
      iamClient: this.clients.IAM,
    });
    this.cleanUpFunctions.push(pipelineExecutionRolePolicyCleanUp);
    // Wait for the role to be ready. If the role is used immediately,
    // the pipeline will fail.
    await wait(5);

    await this.logger.log(MESSAGES.policyAttached);

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingPipeline.replace(
        "${PIPELINE_NAME}",
        this.names.SAGE_MAKER_PIPELINE,
      ),
    );

    // Create the SageMaker pipeline.
    const { cleanUp: pipelineCleanUp } = await createSagemakerPipeline({
      roleArn: pipelineExecutionRoleArn,
      functionArn: lambdaArn,
      sagemakerClient: this.clients.SageMaker,
      name: this.names.SAGE_MAKER_PIPELINE,
    });
    this.cleanUpFunctions.push(pipelineCleanUp);

    await this.logger.log(
      MESSAGES.pipelineCreated.replace(
        "${PIPELINE_NAME}",
        this.names.SAGE_MAKER_PIPELINE,
```

```
    ),
  );

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.creatingS3Bucket.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
  );

  // Create an S3 bucket for storing inputs and outputs.
  const { cleanUp: s3BucketCleanUp } = await createS3Bucket({
    name: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
  });
  this.cleanUpFunctions.push(s3BucketCleanUp);

  await this.logger.log(
    MESSAGES.s3BucketCreated.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
  );

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.uploadingInputData.replace(
      "${BUCKET_NAME}",
      this.names.S3_BUCKET,
    ),
  );

  // Upload CSV Lat/Long data to S3.
  await uploadCSVDataToS3({
    bucketName: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
  });

  await this.logger.log(MESSAGES.inputDataUploaded);

  this.logger.logSeparator();

  await this.prompter.checkContinue(MESSAGES.executePipeline);

  // Execute the SageMaker pipeline.
  const { arn: pipelineExecutionArn } = await startPipelineExecution({
    name: this.names.SAGE_MAKER_PIPELINE,
```

```
      sagemakerClient: this.clients.SageMaker,
      roleArn: pipelineExecutionRoleArn,
      bucketName: this.names.S3_BUCKET,
      queueUrl,
    });

    // Wait for the pipeline execution to finish.
    await waitForPipelineComplete({
      arn: pipelineExecutionArn,
      sagemakerClient: this.clients.SageMaker,
      wait,
    });

    this.logger.logSeparator();

    await this.logger.log(MESSAGES.outputDelay);

    // The getOutput function will throw an error if the output is not
    // found. The retry function will retry a failed function call once
    // ever 10 seconds for 2 minutes.
    const output = await retry({ intervalInMs: 10000, maxRetries: 12 }, () =>
      getObject({
        bucket: this.names.S3_BUCKET,
        s3Client: this.clients.S3,
      }),
    );

    this.logger.logSeparator();
    await this.logger.log(MESSAGES.outputDataRetrieved);
    console.log(output.split("\n").slice(0, 6).join("\n"));
  }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecução
  - StartPipelineExecução
  - UpdatePipeline

Exemplos de Secrets Manager usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Secrets Manager.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Ações

`GetSecretValue`

O código de exemplo a seguir mostra como usar `GetSecretValue`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import {
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";

export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
```

```
  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: secretName,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
  //   CreatedDate: 2023-08-08T19:29:51.294Z,
  //   Name: 'binary-secret-3873048',
  //   SecretBinary: Uint8Array(11) [
  //      98, 105, 110, 97, 114,
  //     121,  32, 100, 97, 116,
  //      97
  //   ],
  //   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
  //   VersionStages: [ 'AWSCURRENT' ]
  // }

  if (response.SecretString) {
    return response.SecretString;
  }

  if (response.SecretBinary) {
    return response.SecretBinary;
  }
};
```

- Para obter detalhes da API, consulte GetSecretValor na Referência AWS SDK for JavaScript da API.

Exemplos do Amazon SES usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon SES.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

Ações

`CreateReceiptFilter`

O código de exemplo a seguir mostra como usar `CreateReceiptFilter`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import {
  CreateReceiptFilterCommand,
  ReceiptFilterPolicy,
} from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
```

```
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
 address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
 addressesOptions.
      },
      /*
        The name of the IP address filter. Only ASCII letters, numbers, underscores,
 or dashes.
        Must be less than 64 characters and start and end with a letter or number.
       */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });

  try {
    return await sesClient.send(createReceiptFilterCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Para obter detalhes da API, consulte CreateReceiptFiltrar na Referência AWS SDK for JavaScript da API.

`CreateReceiptRule`

O código de exemplo a seguir mostra como usar `CreateReceiptRule`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateReceiptRuleCommand, TlsPolicy } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
```

```
  });
};

const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};
```

- Para obter detalhes da API, consulte CreateReceiptRegra na Referência AWS SDK for JavaScript da API.

`CreateReceiptRuleSet`

O código de exemplo a seguir mostra como usar `CreateReceiptRuleSet`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
```

```
const createCreateReceiptRuleSetCommand = (ruleSetName) => {
  return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
};

const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte CreateReceiptRuleSeta Referência AWS SDK for JavaScript da API.

`CreateTemplate`

O código de exemplo a seguir mostra como usar `CreateTemplate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
```

```
    /**
     * The template feature in Amazon SES is based on the Handlebars template
 system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte CreateTemplatea Referência AWS SDK for JavaScript da API.

`DeleteIdentity`

O código de exemplo a seguir mostra como usar `DeleteIdentity`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte DeleteIdentitya Referência AWS SDK for JavaScript da API.

`DeleteReceiptFilter`

O código de exemplo a seguir mostra como usar `DeleteReceiptFilter`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteReceiptFilterCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte DeleteReceiptFiltrar na Referência AWS SDK for JavaScript da API.

`DeleteReceiptRule`

O código de exemplo a seguir mostra como usar `DeleteReceiptRule`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteReceiptRuleCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleCommand = () => {
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte DeleteReceiptRegra na Referência AWS SDK for JavaScript da API.

`DeleteReceiptRuleSet`

O código de exemplo a seguir mostra como usar `DeleteReceiptRuleSet`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};

const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte DeleteReceiptRuleSeta Referência AWS SDK for JavaScript da API.

`DeleteTemplate`

O código de exemplo a seguir mostra como usar `DeleteTemplate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte DeleteTemplatea Referência AWS SDK for JavaScript da API.

`GetTemplate`

O código de exemplo a seguir mostra como usar `GetTemplate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Para obter detalhes da API, consulte GetTemplatea Referência AWS SDK for JavaScript da API.

`ListIdentities`

O código de exemplo a seguir mostra como usar `ListIdentities`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte ListIdentitiesa Referência AWS SDK for JavaScript da API.

`ListReceiptFilters`

O código de exemplo a seguir mostra como usar `ListReceiptFilters`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { ListReceiptFiltersCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
};
```

- Para obter detalhes da API, consulte ListReceiptFiltros na Referência AWS SDK for JavaScript da API.

`ListTemplates`

O código de exemplo a seguir mostra como usar `ListTemplates`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
```

```
  }
};
```

- Para obter detalhes da API, consulte ListTemplatesa Referência AWS SDK for JavaScript da API.

`SendBulkTemplatedEmail`

O código de exemplo a seguir mostra como usar `SendBulkTemplatedEmail`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];
```

```
/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map
 each user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
```

```
    }
    throw caught;
  }
};
```

- Para obter detalhes da API, consulte SendBulkTemplatedEmaila Referência AWS SDK for JavaScript da API.

`SendEmail`

O código de exemplo a seguir mostra como usar `SendEmail`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
```

```
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Para obter detalhes da API, consulte SendEmaila Referência AWS SDK for JavaScript da API.

`SendRawEmail`

O código de exemplo a seguir mostra como usar `SendRawEmail`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Use nodemailer para enviar um e-mail com anexo.

```
import sesClientModule from "@aws-sdk/client-ses";
/**
 * nodemailer wraps the SES SDK and calls SendRawEmail. Use this for more advanced
 * functionality like adding attachments to your email.
 *
 * https://nodemailer.com/transports/ses/
 */
import nodemailer from "nodemailer";

/**
 * @param {string} from An Amazon SES verified email address.
 * @param {*} to An Amazon SES verified email address.
 */
export const sendEmailWithAttachments = (
  from = "from@example.com",
  to = "to@example.com",
) => {
  const ses = new sesClientModule.SESClient({});
  const transporter = nodemailer.createTransport({
    SES: { ses, aws: sesClientModule },
  });

  return new Promise((resolve, reject) => {
    transporter.sendMail(
      {
        from,
        to,
        subject: "Hello World",
        text: "Greetings from Amazon SES!",
```

```
        attachments: [{ content: "Hello World!", filename: "hello.txt" }],
      },
      (err, info) => {
        if (err) {
          reject(err);
        } else {
          resolve(info);
        }
      },
    );
  });
};
```

- Para obter detalhes da API, consulte SendRawE-mail na Referência AWS SDK for JavaScript da API.

`SendTemplatedEmail`

O código de exemplo a seguir mostra como usar `SendTemplatedEmail`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");
```

```
/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
 party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Para obter detalhes da API, consulte SendTemplatedE-mail na Referência AWS SDK for JavaScript da API.

`UpdateTemplate`

O código de exemplo a seguir mostra como usar `UpdateTemplate`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
```

```
    console.log("Failed to update template.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte UpdateTemplatea Referência AWS SDK for JavaScript da API.

`VerifyDomainIdentity`

O código de exemplo a seguir mostra como usar `VerifyDomainIdentity`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();
```

```
  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- Para obter detalhes da API, consulte VerifyDomainIdentidade na Referência AWS SDK for JavaScript da API.

`VerifyEmailIdentity`

O código de exemplo a seguir mostra como usar `VerifyEmailIdentity`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
```

```
    return err;
  }
};
```

- Para obter detalhes da API, consulte VerifyEmailIdentidade na Referência AWS SDK for JavaScript da API.

Exemplos do Amazon SNS usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon SNS.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Amazon SNS

Os exemplos de código a seguir mostram como começar a usar o Amazon SNS.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Inicialize um cliente SNS e liste tópicos em sua conta.

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";
```

```
export const helloSns = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
  const topics = [];

  for await (const page of paginatedTopics) {
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your account.`,
  );
  console.log(topics.map((t) => `  * ${t.TopicArn}`).join("\n"));
};
```

- Para obter detalhes da API, consulte ListTopicsa Referência AWS SDK for JavaScript da API.

Tópicos

- Ações
- Cenários
- Exemplos sem servidor

Ações

`CheckIfPhoneNumberIsOptedOut`

O código de exemplo a seguir mostra como usar `CheckIfPhoneNumberIsOptedOut`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   isOptedOut: false
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CheckIfPhoneNumberIsOptedOut in AWS SDK for JavaScript API Reference.

`ConfirmSubscription`

O código de exemplo a seguir mostra como usar `ConfirmSubscription`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
```

```
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ConfirmSubscriptiona Referência AWS SDK for JavaScript da API.

`CreateTopic`

O código de exemplo a seguir mostra como usar `CreateTopic`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
```

```
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateTopica Referência AWS SDK for JavaScript da API.

`DeleteTopic`

O código de exemplo a seguir mostra como usar `DeleteTopic`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
```

```
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteTopica Referência AWS SDK for JavaScript da API.

`GetSMSAttributes`

O código de exemplo a seguir mostra como usar `GetSMSAttributes`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetSMSAttributes na Referência da API AWS SDK for JavaScript .

`GetTopicAttributes`

O código de exemplo a seguir mostra como usar `GetTopicAttributes`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetTopicAtributos na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Importe o SDK e os módulos do cliente e chame a API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();
```

```
// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetTopicAtributos na Referência AWS SDK for JavaScript da API.

`ListSubscriptions`

O código de exemplo a seguir mostra como usar `ListSubscriptions`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";
```

```
/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListSubscriptionsa Referência AWS SDK for JavaScript da API.

`ListTopics`

O código de exemplo a seguir mostra como usar `ListTopics`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.

- Para obter detalhes da API, consulte ListTopicsa Referência AWS SDK for JavaScript da API.

`Publish`

O código de exemplo a seguir mostra como usar `Publish`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                  if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
```

```
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Publique uma mensagem em um tópico com opções de grupo, duplicação e atributo.

```
  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }
```

```
      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte Publish na Referência da API AWS SDK for JavaScript .

`SetSMSAttributes`

O código de exemplo a seguir mostra como usar `SetSMSAttributes`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
```

```
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para receber detalhes da API, consulte SetSMSAttributes na Referência da API AWS SDK for JavaScript .

`SetTopicAttributes`

O código de exemplo a seguir mostra como usar `SetTopicAttributes`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
```

```
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte SetTopicAtributos na Referência AWS SDK for JavaScript da API.

`Subscribe`

O código de exemplo a seguir mostra como usar `Subscribe`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
```

```
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

Inscreva um aplicativo móvel em um tópico.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 created
 *                            when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Inscreva uma função Lambda em um tópico.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Inscreva uma fila SQS em um tópico.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Inscreva-se com um filtro para um tópico.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
```

```
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute set
 to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte Subscribe na Referência da API AWS SDK for JavaScript .

`Unsubscribe`

O código de exemplo a seguir mostra como usar `Unsubscribe`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente em um módulo separado e exporte-o.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importe o SDK e os módulos do cliente e chame a API.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
```

```
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte Unsubscribe na Referência da API AWS SDK for JavaScript .

Cenários

Publicar mensagens em filas

O exemplo de código a seguir mostra como:

- Criar um tópico (FIFO ou não FIFO).
- Assinar várias filas no tópico com a opção de aplicar um filtro.
- Publicar mensagens no tópico.
- Pesquisar as filas para ver as mensagens recebidas.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Esse é o ponto de entrada para esse fluxo de trabalho.

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";
```

```
export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

O código anterior fornece as dependências necessárias e inicia o fluxo de trabalho. A próxima seção contém a maior parte do exemplo.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
```

```
prompter;

/**
 * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
 * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
 * @param {import('../../libs/prompter.js').Prompter} prompter
 * @param {import('../../libs/logger.js').Logger} logger
 */
constructor(snsClient, sqsClient, prompter, logger) {
  this.snsClient = snsClient;
  this.sqsClient = sqsClient;
  this.prompter = prompter;
  this.logger = logger;
}

async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
```

```
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
```

```
      }),
    );

    const { Attributes } = await this.sqsClient.send(
      new GetQueueAttributesCommand({
        QueueUrl: response.QueueUrl,
        AttributeNames: ["QueueArn"],
      }),
    );

    this.queues.push({
      queueName,
      queueArn: Attributes.QueueArn,
      queueUrl: response.QueueUrl,
    });

    await this.logger.log(
      MESSAGES.queueCreatedNotice
        .replace("${QUEUE_NAME}", queueName)
        .replace("${QUEUE_URL}", response.QueueUrl)
        .replace("${QUEUE_ARN}", Attributes.QueueArn),
    );
  }
}

async attachQueueIamPolicies() {
  for (const [index, queue] of this.queues.entries()) {
    const policy = JSON.stringify(
      {
        Statement: [
          {
            Effect: "Allow",
            Principal: {
              Service: "sns.amazonaws.com",
            },
            Action: "sqs:SendMessage",
            Resource: queue.queueArn,
            Condition: {
              ArnEquals: {
                "aws:SourceArn": this.topicArn,
              },
            },
          },
        ],
```

```
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
```

```
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }
```

```
  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
```

```
                DataType: "String.Array",
                StringValue: JSON.stringify(choices),
              },
            },
          }
        : {}),
    }),
  );

  const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
  });

  if (publishAnother) {
    await this.publishMessages();
  }
}

async receiveAndDeleteMessages() {
  for (const queue of this.queues) {
    const { Messages } = await this.sqsClient.send(
      new ReceiveMessageCommand({
        QueueUrl: queue.queueUrl,
      }),
    );

    if (Messages) {
      await this.logger.log(
        MESSAGES.messagesReceivedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
      console.log(Messages);

      await this.sqsClient.send(
        new DeleteMessageBatchCommand({
          QueueUrl: queue.queueUrl,
          Entries: Messages.map((message) => ({
            Id: message.MessageId,
            ReceiptHandle: message.ReceiptHandle,
          })),
        }),
      );
```

```
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
```

```
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAtributos
  - Publicar
  - ReceiveMessage
  - SetQueueAtributos
  - Assinar
  - Cancelar assinatura

Exemplos sem servidor

Invocar uma função do Lambda em um acionador do Amazon SNS

O exemplo de código a seguir mostra 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.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento do SNS com o JavaScript Lambda usando.

```
// 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;
  }
}
```

Consumindo um evento do SNS com o TypeScript Lambda usando.

```
// 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;
  }
}
```

Exemplos do Amazon SQS usando SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon SQS.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá, Amazon SQS

Os exemplos de código a seguir mostram como começar a usar o Amazon SQS.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Inicializar um cliente Amazon SQS e listar as filas.

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- Para obter detalhes da API, consulte ListQueuesa Referência AWS SDK for JavaScript da API.

Tópicos

Ações

`ChangeMessageVisibility`

O código de exemplo a seguir mostra como usar `ChangeMessageVisibility`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Receba uma mensagem do Amazon SQS e altere sua visibilidade de tempo limite.

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
```

```
      WaitTimeSeconds: 1,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte ChangeMessageVisibilidade na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Receba uma mensagem do Amazon SQS e altere sua visibilidade de tempo limite.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
```

```
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ChangeMessageVisibilidade na Referência AWS SDK for JavaScript da API.

`CreateQueue`

O código de exemplo a seguir mostra como usar `CreateQueue`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie uma fila padrão do Amazon SQS.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Crie uma fila do Amazon SQS com sondagem longa.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: {
```

```
        // When the wait time for the ReceiveMessage API action is greater than 0,
        // long polling is in effect. The maximum long polling wait time is 20
        // seconds. Long polling helps reduce the cost of using Amazon SQS by,
        // eliminating the number of empty responses and false empty responses.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        ReceiveMessageWaitTimeSeconds: "20",
      },
    }),
  );
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateQueuea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Crie uma fila padrão do Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
```

```
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

Crie uma fila do Amazon SQS que aguarda a chegada de uma mensagem.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte CreateQueuea Referência AWS SDK for JavaScript da API.

`DeleteMessage`

O código de exemplo a seguir mostra como usar `DeleteMessage`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Receber e excluir mensagens do Amazon SQS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }
```

```
  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Para obter detalhes da API, consulte DeleteMessagea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Receber e excluir mensagens do Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });
```

```
var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteMessagea Referência AWS SDK for JavaScript da API.

`DeleteMessageBatch`

O código de exemplo a seguir mostra como usar `DeleteMessageBatch`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
```

```
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Para obter detalhes da API, consulte DeleteMessageBatch in AWS SDK for JavaScript API Reference.

`DeleteQueue`

O código de exemplo a seguir mostra como usar `DeleteQueue`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Excluir uma fila do Amazon SQS.

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteQueuea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Excluir uma fila do Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteQueuea Referência AWS SDK for JavaScript da API.

`GetQueueAttributes`

O código de exemplo a seguir mostra como usar `GetQueueAttributes`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
```

```
};
```

- Para obter detalhes da API, consulte GetQueueAtributos na Referência AWS SDK for JavaScript da API.

`GetQueueUrl`

O código de exemplo a seguir mostra como usar `GetQueueUrl`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha o URL para uma fila do Amazon SQS.

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetQueueUrl na Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Obtenha o URL para uma fila do Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte GetQueueUrl na Referência AWS SDK for JavaScript da API.

`ListQueues`

O código de exemplo a seguir mostra como usar `ListQueues`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Listar filas do Amazon SQS.

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    urls.forEach((url) => console.log(url));
  }

  return urls;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListQueuesa Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Listar filas do Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListQueuesa Referência AWS SDK for JavaScript da API.

`ReceiveMessage`

O código de exemplo a seguir mostra como usar `ReceiveMessage`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Receba uma mensagem de uma fila do Amazon SQS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
```

```
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

Receba uma mensagem de uma fila do Amazon SQS usando o suporte de sondagem longa.

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
    // The duration (in seconds) for which the call waits for a message
    // to arrive in the queue before returning. If a message is available,
    // the call returns sooner than WaitTimeSeconds. If no messages are
    // available and the wait time expires, the call returns successfully
    // with an empty list of messages.
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
    WaitTimeSeconds: 20,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte ReceiveMessagea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Receba uma mensagem de uma fila do Amazon SQS usando o suporte de sondagem longa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ReceiveMessagea Referência AWS SDK for JavaScript da API.

`SendMessage`

O código de exemplo a seguir mostra como usar `SendMessage`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Enviar uma mensagem para uma fila do Amazon SQS.

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
    MessageBody:
      "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte SendMessagea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Enviar uma mensagem para uma fila do Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of 12/11/2016.",
```

```
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte SendMessagea Referência AWS SDK for JavaScript da API.

`SetQueueAttributes`

O código de exemplo a seguir mostra como usar `SetQueueAttributes`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
```

```
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Configurar uma fila do Amazon SQS para usar sondagem longa.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Configurar uma dead-letter queue.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
```

```
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obter detalhes da API, consulte SetQueueAtributos na Referência AWS SDK for JavaScript da API.

Cenários

Publicar mensagens em filas

O exemplo de código a seguir mostra como:

- Criar um tópico (FIFO ou não FIFO).
- Assinar várias filas no tópico com a opção de aplicar um filtro.
- Publicar mensagens no tópico.
- Pesquisar as filas para ver as mensagens recebidas.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Esse é o ponto de entrada para esse fluxo de trabalho.

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

O código anterior fornece as dependências necessárias e inicia o fluxo de trabalho. A próxima seção contém a maior parte do exemplo.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
```

```
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../../libs/prompter.js').Prompter} prompter
   * @param {import('../../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
    await this.logger.log(MESSAGES.snsFifoDescription);
    this.isFifo = await this.prompter.confirm({
      message: MESSAGES.snsFifoPrompt,
    });

    if (this.isFifo) {
      this.logger.logSeparator(MESSAGES.headerDedup);
      await this.logger.log(MESSAGES.deduplicationNotice);
      await this.logger.log(MESSAGES.deduplicationDescription);
      this.autoDedup = await this.prompter.confirm({
        message: MESSAGES.deduplicationPrompt,
      });
    }
  }

  async createTopic() {
```

```
    await this.logger.log(MESSAGES.creatingTopics);
    this.topicName = await this.prompter.input({
      message: MESSAGES.topicNamePrompt,
    });
    if (this.isFifo) {
      this.topicName += ".fifo";
      this.logger.logSeparator(MESSAGES.headerFifoNaming);
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
```

```
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
```

```
            Resource: queue.queueArn,
            Condition: {
              ArnEquals: {
                "aws:SourceArn": this.topicArn,
              },
            },
          },
        ],
      },
      null,
      2,
    );

    if (index !== 0) {
      this.logger.logSeparator();
    }

    await this.logger.log(MESSAGES.attachPolicyNotice);
    console.log(policy);
    const addPolicy = await this.prompter.confirm({
      message: MESSAGES.addPolicyConfirmation.replace(
        "${QUEUE_NAME}",
        queue.queueName,
      ),
    });

    if (addPolicy) {
      await this.sqsClient.send(
        new SetQueueAttributesCommand({
          QueueUrl: queue.queueUrl,
          Attributes: {
            Policy: policy,
          },
        }),
      );
      queue.policy = policy;
    } else {
      await this.logger.log(
        MESSAGES.policyNotAttachedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
    }
```

```
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
```

```
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
```

```
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
```

```
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
```

```
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAtributos

- Publicar
- ReceiveMessage
- SetQueueAtributos
- Assinar
- Cancelar assinatura

Exemplos sem servidor

Invocar uma função do Lambda em um trigger do Amazon SQS

O exemplo de código a seguir mostra 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.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumindo um evento SQS com o JavaScript Lambda usando.

```
// 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;
  }
}
```

Consumindo um evento SQS com o TypeScript Lambda usando.

```
// 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;
  }
}
```

Relatar falhas de itens em lote para funções do Lambda com um trigger do Amazon SQS

O exemplo de código a seguir mostra 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.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatando falhas de itens em lote do SQS com o uso do JavaScript Lambda.

```
// 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}`);
}
```

Relatando falhas de itens em lote do SQS com o uso do TypeScript Lambda.

```
// 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}`);
}
```

Exemplos de Step Functions usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com Step Functions.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`StartExecution`

O código de exemplo a seguir mostra como usar `StartExecution`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

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

import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
```

```
  // }
  return response;
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- Para obter detalhes da API, consulte StartExecutiona Referência AWS SDK for JavaScript da API.

AWS STS exemplos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com AWS STS.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`AssumeRole`

O código de exemplo a seguir mostra como usar `AssumeRole`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Assuma um perfil do IAM.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
```

```
  }
};
```

- Para obter detalhes da API, consulte AssumeRolea Referência AWS SDK for JavaScript da API.

SDK para JavaScript (v2)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
const AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

var roleToAssume = {
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",
  RoleSessionName: "session1",
  DurationSeconds: 900,
};
var roleCreds;

// Create the STS service object
var sts = new AWS.STS({ apiVersion: "2011-06-15" });

//Assume Role
sts.assumeRole(roleToAssume, function (err, data) {
  if (err) console.log(err, err.stack);
  else {
    roleCreds = {
      accessKeyId: data.Credentials.AccessKeyId,
      secretAccessKey: data.Credentials.SecretAccessKey,
      sessionToken: data.Credentials.SessionToken,
    };
    stsGetCallerIdentity(roleCreds);
  }
});
```

```
//Get Arn of current identity
function stsGetCallerIdentity(creds) {
  var stsParams = { credentials: creds };
  // Create STS service object
  var sts = new AWS.STS(stsParams);

  sts.getCallerIdentity({}, function (err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data.Arn);
    }
  });
}
```

- Para obter detalhes da API, consulte AssumeRolea Referência AWS SDK for JavaScript da API.

AWS Support exemplos usando o SDK para JavaScript (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com AWS Support.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Conceitos básicos

Olá AWS Support

O exemplo de código a seguir mostra como começar a usar o AWS Support.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Invoque `main()` para executar o exemplo.

```
import {
  DescribeServicesCommand,
  SupportClient,
} from "@aws-sdk/client-support";

// Change the value of 'region' to your preferred AWS Region.
const client = new SupportClient({ region: "us-east-1" });

const getServiceCount = async () => {
  try {
    const { services } = await client.send(new DescribeServicesCommand({}));
    return services.length;
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
    } else {
      throw err;
    }
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- Para obter detalhes da API, consulte DescribeServicesa Referência AWS SDK for JavaScript da API.

Tópicos

Ações

`AddAttachmentsToSet`

O código de exemplo a seguir mostra como usar `AddAttachmentsToSet`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { AddAttachmentsToSetCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new attachment set or add attachments to an existing set.
    // Provide an 'attachmentSetId' value to add attachments to an existing set.
    // Use AddCommunicationToCase or CreateCase to associate an attachment set with
 a support case.
    const response = await client.send(
      new AddAttachmentsToSetCommand({
        // You can add up to three attachments per set. The size limit is 5 MB per
 attachment.
        attachments: [
          {
            fileName: "example.txt",
            data: new TextEncoder().encode("some example text"),
          },
```

```
        ],
      }),
    );
    // Use this ID in AddCommunicationToCase or CreateCase.
    console.log(response.attachmentSetId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte AddAttachmentsToSeta Referência AWS SDK for JavaScript da API.

`AddCommunicationToCase`

O código de exemplo a seguir mostra como usar `AddCommunicationToCase`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { AddCommunicationToCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  let attachmentSetId;

  try {
    // Add a communication to a case.
    const response = await client.send(
      new AddCommunicationToCaseCommand({
        communicationBody: "Adding an attachment.",
        // Set value to an existing support case id.
        caseId: "CASE_ID",
```

```
        // Optional. Set value to an existing attachment set id to add attachments
 to the case.
        attachmentSetId,
      }),
    );
    console.log(response);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte AddCommunicationToCasea Referência AWS SDK for JavaScript da API.

`CreateCase`

O código de exemplo a seguir mostra como usar `CreateCase`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { CreateCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new case and log the case id.
    // Important: This creates a real support case in your account.
    const response = await client.send(
      new CreateCaseCommand({
        // The subject line of the case.
        subject: "IGNORE: Test case",
        // Use DescribeServices to find available service codes for each service.
```

```
        serviceCode: "service-quicksight-end-user",
        // Use DescribeSecurityLevels to find available severity codes for your
 support plan.
        severityCode: "low",
        // Use DescribeServices to find available category codes for each service.
        categoryCode: "end-user-support",
        // The main description of the support case.
        communicationBody: "This is a test. Please ignore.",
      }),
    );
    console.log(response.caseId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte CreateCasea Referência AWS SDK for JavaScript da API.

`DescribeAttachment`

O código de exemplo a seguir mostra como usar `DescribeAttachment`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeAttachmentCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the metadata and content of an attachment.
    const response = await client.send(
      new DescribeAttachmentCommand({
        // Set value to an existing attachment id.
```

```
        // Use DescribeCommunications or DescribeCases to find an attachment id.
        attachmentId: "ATTACHMENT_ID",
      }),
    );
    console.log(response.attachment?.fileName);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeAttachmenta Referência AWS SDK for JavaScript da API.

`DescribeCases`

O código de exemplo a seguir mostra como usar `DescribeCases`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeCasesCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all of the unresolved cases in your account.
    // Filter or expand results by providing parameters to the DescribeCasesCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecasescommandinput.html
    const response = await client.send(new DescribeCasesCommand({}));
```

```
    const caseIds = response.cases.map((supportCase) => supportCase.caseId);
    console.log(caseIds);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeCasesa Referência AWS SDK for JavaScript da API.

`DescribeCommunications`

O código de exemplo a seguir mostra como usar `DescribeCommunications`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeCommunicationsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all communications for the support case.
    // Filter results by providing parameters to the DescribeCommunicationsCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecommunicationscommandinput.html
    const response = await client.send(
      new DescribeCommunicationsCommand({
        // Set value to an existing case id.
        caseId: "CASE_ID",
```

```
      }),
    );
    const text = response.communications.map((item) => item.body).join("\n");
    console.log(text);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeCommunicationsa Referência AWS SDK for JavaScript da API.

`DescribeSeverityLevels`

O código de exemplo a seguir mostra como usar `DescribeSeverityLevels`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { DescribeSeverityLevelsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the list of severity levels.
    // The available values depend on the support plan for the account.
    const response = await client.send(new DescribeSeverityLevelsCommand({}));
    console.log(response.severityLevels);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte DescribeSeverityNíveis na Referência AWS SDK for JavaScript da API.

`ResolveCase`

O código de exemplo a seguir mostra como usar `ResolveCase`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

```
import { ResolveCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

const main = async () => {
  try {
    const response = await client.send(
      new ResolveCaseCommand({
        caseId: "CASE_ID",
      }),
    );

    console.log(response.finalCaseStatus);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte ResolveCasea Referência AWS SDK for JavaScript da API.

Cenários

Conceitos básicos de casos

O exemplo de código a seguir mostra como:

- Obter e exibir os serviços disponíveis e os níveis de gravidade dos casos.
- Criar um caso de suporte usando um serviço, uma categoria e um nível de gravidade selecionados.
- Obter e exibir uma lista de casos em aberto para o dia atual.
- Adicionar um conjunto de anexos e uma comunicação ao novo caso.
- Descrever o novo anexo e a comunicação para o caso.
- Resolver o caso.
- Obter e exibir uma lista de casos resolvidos para o dia atual.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Execute um cenário interativo no terminal.

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import * as inquirer from "@inquirer/prompts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
```

```
const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};

const client = new SupportClient({ region: "us-east-1" });

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});

  try {
    await client.send(command);
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
    } else {
      throw err;
    }
  }
};

/**
 * Select a service from the list returned from DescribeServices.
 */
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const selectedService = await inquirer.select({
    message:
      "Select a service. Your support case will be created for this service. The
 list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
};

/**
 * @param {{ categories: import('@aws-sdk/client-support').Category[]}} service
 */
export const getCategory = async (service) => {
  const selectedCategory = await inquirer.select({
```

```
    message: "Select a category.",
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const selectedSeverityLevel = await inquirer.select({
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

/**
 * Create a new support case
 * @param {{
 *  selectedService: import('@aws-sdk/client-support').Service
 *  selectedCategory: import('@aws-sdk/client-support').Category
 *  selectedSeverityLevel: import('@aws-sdk/client-support').SeverityLevel
 * }} selections
 * @returns
 */
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodaysOpenCases = async () => {
```

```
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });

  const { cases } = await client.send(command);

  if (cases.length === 0) {
    throw new Error(
      "Unexpected number of cases. Expected more than 0 open cases.",
    );
  }
  return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
  const command = new AddAttachmentsToSetCommand({
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
```

```
  });
  const { communications } = await client.send(command);
  return communications;
};

/**
 * @param {import('@aws-sdk/client-support').Communication[]} communications
 */
export const getFirstAttachment = (communications) => {
  const firstCommWithAttachment = communications.find(
    (c) => c.attachmentSet.length > 0,
  );
  return firstCommWithAttachment?.attachmentSet[0].attachmentId;
};

// Get an attachment.
export const getAttachment = async (attachmentId) => {
  const command = new DescribeAttachmentCommand({
    attachmentId,
  });
  const { attachment } = await client.send(command);
  return attachment;
};

// Resolve the case matching the given case ID.
export const resolveCase = async (caseId) => {
  const shouldResolve = await inquirer.confirm({
    message: `Do you want to resolve ${caseId}?`,
  });

  if (shouldResolve) {
    const command = new ResolveCaseCommand({
      caseId: caseId,
    });

    await client.send(command);
    return true;
  }
  return false;
};

/**
 * Find a specific case in the list of provided cases by case ID.
 * If the case is not found, and the results are paginated, continue
```

```
 * paging through the results.
 * @param {{
 *   caseId: string,
 *   cases: import('@aws-sdk/client-support').CaseDetails[]
 *   nextToken: string
 * }} options
 * @returns
 */
export const findCase = async ({ caseId, cases, nextToken }) => {
  const foundCase = cases.find((c) => c.caseId === caseId);

  if (foundCase) {
    return foundCase;
  }

  if (nextToken) {
    const response = await client.send(
      new DescribeCasesCommand({
        nextToken,
        includeResolvedCases: true,
      }),
    );
    return findCase({
      caseId,
      cases: response.cases,
      nextToken: response.nextToken,
    });
  }

  throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
```

```
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
 select one.
    const selectedCategory = await getCategory(selectedService);

    // Provide the severity available severity levels for the account and prompt the
 user to select one.
    const selectedSeverityLevel = await getSeverityLevel();

    // Create a support case.
    console.log("\nCreating a support case.");
    caseId = await createCase({
      selectedService,
      selectedCategory,
      selectedSeverityLevel,
    });
    console.log(`Support case created: ${caseId}`);

    // Display a list of open support cases created today.
    const todaysOpenCases = await retry(
      { intervalInMs: 1000, maxRetries: 15 },
      getTodaysOpenCases,
    );
    console.log(
      `\nOpen support cases created today: ${todaysOpenCases.length}`,
    );
    console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

    // Create an attachment set.
    console.log("\nCreating an attachment set.");
    const attachmentSetId = await createAttachmentSet();
    console.log(`Attachment set created: ${attachmentSetId}`);
```

```
  // Add the attachment set to the support case.
  console.log(`\nAdding attachment set to ${caseId}`);
  await linkAttachmentSetToCase(attachmentSetId, caseId);
  console.log(`Attachment set added to ${caseId}`);

  // List the communications for a support case.
  console.log(`\nListing communications for ${caseId}`);
  const communications = await getCommunications(caseId);
  console.log(
    communications
      .map(
        (c) =>
          `Communication created on ${c.timeCreated}. Has
${c.attachmentSet.length} attachments.`,
      )
      .join("\n"),
  );

  // Describe the first attachment.
  console.log(`\nDescribing attachment ${attachmentSetId}`);
  const attachmentId = getFirstAttachment(communications);
  const attachment = await getAttachment(attachmentId);
  console.log(
    `Attachment is the file '${
      attachment.fileName
    }' with data: \n${new TextDecoder().decode(attachment.data)}`,
  );

  // Confirm that the support case should be resolved.
  const isResolved = await resolveCase(caseId);
  if (isResolved) {
    // List the resolved cases and include the one previously created.
    // Resolved cases can take a while to appear.
    console.log(
      "\nWaiting for case status to be marked as resolved. This can take some
time.",
    );
    const resolvedCases = await retry(
      { intervalInMs: 20000, maxRetries: 15 },
      () => getTodaysResolvedCases(caseId),
    );
    console.log("Resolved cases:");
    console.log(resolvedCases.map((c) => c.caseId).join("\n"));
```

```
    }
  } catch (err) {
    console.error(err);
  }
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript .
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases
  - DescribeCommunications
  - DescribeServices
  - DescribeSeverityNíveis
  - ResolveCase

Exemplos do Amazon Transcribe usando SDK JavaScript para (v3)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK for JavaScript (v3) com o Amazon Transcribe.

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.

Cada exemplo inclui um link para GitHub, onde você pode encontrar instruções sobre como configurar e executar o código no contexto.

Tópicos

- Ações

Ações

`DeleteMedicalTranscriptionJob`

O código de exemplo a seguir mostra como usar `DeleteMedicalTranscriptionJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Excluir um trabalho de transcrição médica.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
```

```
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteMedicalTranscriptionJoba Referência AWS SDK for JavaScript da API.

`DeleteTranscriptionJob`

O código de exemplo a seguir mostra como usar `DeleteTranscriptionJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Excluir um trabalho de transcrição.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
```

```
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Crie o cliente.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte DeleteTranscriptionJob in AWS SDK for JavaScript API Reference.

`ListMedicalTranscriptionJobs`

O código de exemplo a seguir mostra como usar `ListMedicalTranscriptionJobs`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
```

```
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Listar trabalhos de transcrição médica.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListMedicalTranscriptionJobsa Referência AWS SDK for JavaScript da API.

`ListTranscriptionJobs`

O código de exemplo a seguir mostra como usar `ListTranscriptionJobs`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Listar trabalhos de transcrição.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Crie o cliente.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte ListTranscriptionJobs in AWS SDK for JavaScript API Reference.

`StartMedicalTranscriptionJob`

O código de exemplo a seguir mostra como usar `StartMedicalTranscriptionJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie o cliente.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Iniciar um trabalho de transcrição médica.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte StartMedicalTranscriptionJoba Referência AWS SDK for JavaScript da API.

`StartTranscriptionJob`

O código de exemplo a seguir mostra como usar `StartTranscriptionJob`.

SDK para JavaScript (v3)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e veja como configurar e executar no AWS Code Examples Repository.

Iniciar um trabalho de transcrição.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/
hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Crie o cliente.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Para obter mais informações, consulte o Guia do desenvolvedor do AWS SDK for JavaScript.
- Para obter detalhes da API, consulte StartTranscriptionJob in AWS SDK for JavaScript API Reference.

Exemplos de serviços cruzados usando o SDK para JavaScript (v3)

Os aplicativos de exemplo a seguir usam o AWS SDK for JavaScript (v3) para trabalhar em vários Serviços da AWS.

Os exemplos de serviços cruzados visam um nível avançado de experiência para ajudar a começar a criar aplicativos.

Exemplos

- Criar uma aplicação Amazon Transcribe
- Criar uma aplicação de transmissão do Amazon Transcribe
- Criar uma aplicação para enviar dados para uma tabela do DynamoDB
- Crie um chatbot do Amazon Lex para engajar os visitantes do site
- Criar uma aplicação de gerenciamento de ativos de fotos que permita que os usuários gerenciem fotos usando rótulos
- Criar uma aplicação Web para monitorar dados do DynamoDB
- Crie um rastreador de itens de trabalho do Aurora Sem Servidor
- Criar uma aplicação de exploração do Amazon Textract
- Criar uma aplicação que analise o feedback dos clientes e sintetize o áudio
- Detecte PPE em imagens com o Amazon Rekognition usando um SDK AWS
- Detecte objetos em imagens com o Amazon Rekognition usando um SDK AWS
- Detecte pessoas e objetos em um vídeo com o Amazon Rekognition usando um SDK AWS

- Invocar uma função do Lambda em um navegador
- Usar o API Gateway para invocar uma função do Lambda
- Usar Step Functions para invocar funções do Lambda
- Usar eventos programados para chamar uma função do Lambda

Criar uma aplicação Amazon Transcribe

SDK para JavaScript (v3)

Crie uma aplicação que use o Amazon Transcribe para transcrever e exibir gravações de voz no navegador. A aplicação usa dois buckets do Amazon Simple Storage Service (Amazon S3), um para hospedar o código da aplicação e outro para armazenar transcrições. A aplicação usa um grupo de usuários do Amazon Cognito para autenticar seus usuários. Os usuários autenticados têm permissões AWS Identity and Access Management (IAM) para acessar os AWS serviços necessários.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Esse exemplo também está disponível no Guia do desenvolvedor do AWS SDK for JavaScript v3.

Serviços utilizados neste exemplo

- Identidade do Amazon Cognito
- Amazon S3
- Amazon Transcribe

Criar uma aplicação de transmissão do Amazon Transcribe

SDK para JavaScript (v3)

Mostra como usar o Amazon Transcribe para construir uma aplicação que registra, transcreve e traduz áudio ao vivo em tempo real, e envia os resultados por e-mail usando o Amazon Simple Email Service (Amazon SES).

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços utilizados neste exemplo

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Criar uma aplicação para enviar dados para uma tabela do DynamoDB

SDK para JavaScript (v3)

Este exemplo mostra como criar uma aplicação que permite que os usuários enviem dados para uma tabela do Amazon DynamoDB e enviem uma mensagem de texto ao administrador usando o Amazon Simple Notification Service (Amazon SNS).

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Esse exemplo também está disponível no Guia do desenvolvedor do AWS SDK for JavaScript v3.

Serviços usados neste exemplo

- DynamoDB
- Amazon SNS

Crie um chatbot do Amazon Lex para engajar os visitantes do site

SDK para JavaScript (v3)

Mostra como usar a API do Amazon Lex para criar um Chatbot em um aplicativo da web para engajar os visitantes do site.

Para obter o código-fonte completo e instruções sobre como configurar e executar, consulte o exemplo completo Criando um chatbot Amazon Lex no guia do AWS SDK for JavaScript desenvolvedor.

Serviços utilizados neste exemplo

- Amazon Comprehend
- Amazon Lex

- Amazon Translate

Criar uma aplicação de gerenciamento de ativos de fotos que permita que os usuários gerenciem fotos usando rótulos

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, veja o exemplo completo em 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

Criar uma aplicação Web para monitorar dados do DynamoDB

SDK para JavaScript (v3)

Mostra como usar a API do Amazon DynamoDB para construir uma aplicação Web dinâmica que monitora os dados de trabalho do DynamoDB.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon SES

Crie um rastreador de itens de trabalho do Aurora Sem Servidor

SDK para JavaScript (v3)

Mostra como usar o AWS SDK for JavaScript (v3) para criar um aplicativo web que rastreia itens de trabalho em um banco de dados Amazon Aurora e envia relatórios por e-mail usando o Amazon Simple Email Service (Amazon SES). Este exemplo usa um front-end criado com React.js para interagir com um back-end Node.js Express.

- Integre um aplicativo web React.js com Serviços da AWS o.
- Liste, adicione e atualize itens em uma tabela do Aurora.
- Use o Amazon SES para enviar um relatório por e-mail dos itens de trabalho filtrados.
- Implante e gerencie recursos de exemplo com o AWS CloudFormation script incluído.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços utilizados neste exemplo

- Aurora
- Amazon RDS
- Serviços de dados do Amazon RDS
- Amazon SES

Criar uma aplicação de exploração do Amazon Textract

SDK para JavaScript (v3)

Mostra como usar o AWS SDK for JavaScript para criar um aplicativo React que usa o Amazon Textract para extrair dados de uma imagem de documento e exibi-los em uma página da web interativa. Este exemplo é executado em um navegador da Web e requer uma identidade autenticada do Amazon Cognito como credenciais. Ele usa o Amazon Simple Storage Service (Amazon S3) para armazenamento e, para notificações, pesquisa uma fila do Amazon Simple Queue Service (Amazon SQS) que está inscrita em um tópico do Amazon Simple Notification Service (Amazon SNS).

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços utilizados neste exemplo

- Identidade do Amazon Cognito
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Criar uma aplicação que analise o feedback dos clientes e sintetize o áudio

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

Detecte PPE em imagens com o Amazon Rekognition usando um SDK AWS

SDK para JavaScript (v3)

Mostra como usar o Amazon Rekognition AWS SDK for JavaScript com o para criar um aplicativo para detectar equipamentos de proteção individual (EPI) em imagens localizadas em um bucket do Amazon Simple Storage Service (Amazon S3). A aplicação salva os resultados em uma tabela do Amazon DynamoDB e envia uma notificação por e-mail ao administrador com os resultados usando o Amazon Simple Email Service (Amazon SES).

Aprenda como:

- Criar um usuário não autenticado usando o Amazon Cognito.
- Analisar imagens em busca de EPI usando o Amazon Rekognition.
- Verificar um endereço de e-mail para o Amazon SES.
- Atualizar uma tabela do DynamoDB com resultados.
- Enviar uma notificação por e-mail usando o Amazon SES.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços usados neste exemplo

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Detecte objetos em imagens com o Amazon Rekognition usando um SDK AWS

SDK para JavaScript (v3)

Mostra como usar o Amazon Rekognition AWS SDK for JavaScript com o para criar um aplicativo que usa o Amazon Rekognition para identificar objetos por categoria em imagens localizadas em um bucket do Amazon Simple Storage Service (Amazon S3). A aplicação envia uma notificação por e-mail ao administrador com os resultados usando o Amazon Simple Email Service (Amazon SES).

Aprenda como:

- Criar um usuário não autenticado usando o Amazon Cognito.
- Analisar imagens em busca de objetos usando o Amazon Rekognition.
- Verificar um endereço de e-mail para o Amazon SES.
- Enviar uma notificação por e-mail usando o Amazon SES.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços utilizados neste exemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Detecte pessoas e objetos em um vídeo com o Amazon Rekognition usando um SDK AWS

SDK para JavaScript (v3)

Mostra como usar o Amazon Rekognition AWS SDK for JavaScript com o para criar um aplicativo para detectar faces e objetos em vídeos localizados em um bucket do Amazon Simple Storage Service (Amazon S3). A aplicação envia uma notificação por e-mail ao administrador com os resultados usando o Amazon Simple Email Service (Amazon SES).

Aprenda como:

- Criar um usuário não autenticado usando o Amazon Cognito.
- Analisar imagens em busca de EPI usando o Amazon Rekognition.
- Verificar um endereço de e-mail para o Amazon SES.
- Enviar uma notificação por e-mail usando o Amazon SES.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços utilizados neste exemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Invocar uma função do Lambda em um navegador

SDK para JavaScript (v2)

Você pode criar um aplicativo baseado em navegador que usa uma AWS Lambda função para atualizar uma tabela do Amazon DynamoDB com as seleções do usuário.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços usados neste exemplo

- DynamoDB
- Lambda

SDK para JavaScript (v3)

Você pode criar um aplicativo baseado em navegador que usa uma AWS Lambda função para atualizar uma tabela do Amazon DynamoDB com as seleções do usuário. Este aplicativo usa AWS SDK for JavaScript v3.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em GitHub.

Serviços usados neste exemplo

- DynamoDB

- Lambda

Usar o API Gateway para invocar uma função do Lambda

SDK para JavaScript (v3)

Mostra como criar uma AWS Lambda função usando a API de tempo de JavaScript execução do Lambda. Este exemplo invoca AWS serviços diferentes para realizar 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, veja o exemplo completo em 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

Usar Step Functions para invocar funções do Lambda

SDK para JavaScript (v3)

Mostra como criar um fluxo de trabalho AWS sem servidor usando AWS Step Functions e. AWS SDK for JavaScript Cada etapa do fluxo de trabalho é implementada usando uma AWS Lambda função.

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, veja o exemplo completo em 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

Usar eventos programados para chamar uma função do Lambda

SDK para JavaScript (v3)

Mostra como criar um evento EventBridge programado pela Amazon que invoca uma AWS Lambda função. Configure EventBridge para usar uma expressão cron para agendar quando a função Lambda é invocada. Neste exemplo, você cria uma função Lambda usando a API de tempo de execução do JavaScript Lambda. Este exemplo invoca AWS serviços diferentes para realizar 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, veja o exemplo completo em 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

Segurança para este AWS produto ou serviço

A segurança da nuvem na Amazon Web Services (AWS) é a nossa maior prioridade. Como cliente da AWS , você contará com um data center 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 você AWS e você. O modelo de responsabilidade compartilhada descreve isso como a Segurança da nuvem e a Segurança na nuvem.

Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa todos os serviços oferecidos na AWS nuvem e fornecer serviços que você possa usar com segurança. Nossa responsabilidade de segurança é a maior prioridade em AWS, e a eficácia de nossa segurança é regularmente testada e verificada por auditores terceirizados como parte dos Programas de AWS Conformidade.

Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você está usando e por outros fatores, incluindo a sensibilidade de seus dados, os requisitos da sua organização e as leis e regulamentos aplicáveis.

Esse AWS produto ou serviço segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Tópicos

Proteção de dados neste AWS produto ou serviço

O modelo de responsabilidade AWS compartilhada de se aplica à proteção de dados neste AWS produto ou serviço. Conforme descrito neste modelo, AWS é responsável por proteger a

infraestrutura global que executa todos os 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 ter mais informações sobre a privacidade de dados, consulte as Perguntas frequentes sobre privacidade de dados. Para ter mais informações sobre a proteção de dados na Europa, consulte a AWS postagem do blog Shared Responsibility Model and GDPR no AWS Blog de segurança da.

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com AWS IAM Identity Center ou 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. AWS Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão 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 AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint 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 email dos seus clientes, em marcações ou campos de formato livre, como um campo Name (Nome). Isso inclui quando você trabalha com esse AWS produto, serviço ou outro Serviços da AWS usando o console, a API ou AWS os SDKs. AWS CLI 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, recomendemos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Identity and Access Management

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) a usar AWS os recursos. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Tópicos

Público

A forma como você usa AWS Identity and Access Management (IAM) difere, dependendo do trabalho que você faz AWS.

Usuário do serviço — Se você Serviços da AWS costuma fazer seu trabalho, seu administrador fornece as credenciais e as permissões de que você precisa. À medida que você usa mais AWS recursos para fazer seu trabalho, talvez precise de permissões adicionais. Entender como o acesso é gerenciado pode ajudar você a solicitar as permissões corretas ao seu administrador. Se você não conseguir acessar um recurso no AWS, consulte Solução de problemas AWS de identidade e acesso o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Administrador de serviços — Se você é responsável pelos AWS recursos da sua empresa, provavelmente tem acesso total AWS a. É seu trabalho determinar quais AWS recursos e recursos seus 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 sua empresa pode usar o IAM AWS, consulte o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Administrador do IAM: Se você for um administrador do IAM, talvez queira saber detalhes sobre como pode gravar políticas para gerenciar acesso ao AWS. Para ver exemplos de políticas AWS

baseadas em identidade que você pode usar no IAM, consulte o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Autenticando com identidades

A autenticação é como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado (conectado AWS) como o Usuário raiz da conta da AWS, como usuário do IAM ou assumindo uma função do IAM.

Você pode entrar AWS como uma identidade federada usando credenciais fornecidas por meio de uma fonte de identidade. AWS IAM Identity Center Usuários (IAM Identity Center), a autenticação de login único da sua empresa e suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Ao acessar AWS usando a federação, você está assumindo indiretamente uma função.

Dependendo do tipo de usuário que você é, você pode entrar no AWS Management Console ou no portal de AWS acesso. Para obter mais informações sobre como fazer login AWS, consulte Como fazer login Conta da AWS no Guia do Início de Sessão da AWS usuário.

Se você acessar AWS programaticamente, AWS fornece um kit de desenvolvimento de software (SDK) e uma interface de linha de comando (CLI) para assinar criptograficamente suas solicitações usando suas credenciais. Se você não usa AWS ferramentas, você mesmo deve assinar as solicitações. Para obter mais informações sobre como usar o método recomendado para assinar solicitações por conta própria, consulte Assinatura de solicitações de AWS API no Guia do usuário do IAM.

Independente do método de autenticação usado, também pode ser exigido que você forneça informações adicionais de segurança. Por exemplo, AWS recomenda que você use a autenticação multifator (MFA) para aumentar a segurança da sua conta. Para saber mais, consulte Autenticação Multifator no AWS IAM Identity Center Guia do Usuário. Usar a autenticação multifator (MFA) na AWS no Guia do Usuário do IAM.

Conta da AWS usuário root

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos Serviços da AWS os recursos da conta. Essa identidade é chamada de usuário Conta da AWS raiz e é acessada fazendo login com o endereço de e-mail e a senha que você usou para criar a conta. É altamente recomendável não usar o usuário raiz para tarefas diárias. Proteja as credenciais do usuário raiz e use-as para executar as tarefas que somente ele pode 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 usuários humanos, incluindo usuários que precisam de acesso de administrador, usem a federação com um provedor de identidade para acessar Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório de usuários corporativo, de um provedor de identidade da web AWS Directory Service, do diretório do Identity Center ou de qualquer usuário que acesse usando credenciais fornecidas Serviços da AWS por meio de uma fonte de identidade. Quando as identidades federadas são acessadas Contas da AWS, elas assumem funções, e as funções fornecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos usar o . AWS IAM Identity Center Você pode criar usuários e grupos no IAM Identity Center ou pode se conectar e sincronizar com um conjunto de usuários e grupos em sua própria fonte de identidade para uso em todos os seus Contas da AWS aplicativos. Para obter mais informações sobre o Centro de Identidade do IAM, consulte O que é o Centro de Identidade do IAM? no AWS IAM Identity Center Manual do Usuário do.

Usuários e grupos do IAM

Um usuário do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas para uma única pessoa ou aplicativo. Sempre que possível, recomendamos depender de 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 uso específicos que exijam credenciais de longo prazo com usuários do IAM, recomendamos alternar as chaves de acesso. Para obter mais informações, consulte Alterne Chaves de Acesso Regularmente para Casos de Uso 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 usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um nome de grupo 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 um aplicativo, mas uma função pode ser assumida 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 (Ao Invés de uma Função) no Guia do Usuário do IAM.

Perfis do IAM

Uma função do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. Você pode assumir temporariamente uma função do IAM no AWS Management Console trocando de funções. Você pode assumir uma função chamando uma operação de AWS API AWS CLI ou usando uma URL personalizada. Para obter mais informações sobre métodos para usar perfis, consulte Usando Funções do IAM no Guia do Usuário do IAM.

Funções do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, você pode 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 Criando um Perfil para um Provedor de Identidades Terceirizado no Guia do Usuário do IAM. Se você usa o IAM Identity Center, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o IAM Identity Center correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de Permissões no AWS IAM Identity Center Manual do Usuário.
- Permissões de usuários temporárias do IAM: um usuário ou perfil do IAM pode assumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas: você pode usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) acesse recursos na sua conta de uma conta diferente. As funções são a forma primária de conceder acesso entre contas. No entanto, com alguns Serviços da AWS, você pode anexar uma política diretamente a um recurso (em vez de usar uma função como proxy). Para aprender a diferença entre funções e políticas baseadas em recurso para acesso entre contas, consulte Como as Funções 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 aplicativos no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões de chamada da entidade principal, uma função de serviço ou uma função vinculada ao serviço.

- Sessões de acesso direto (FAS) — Quando você usa um usuário ou uma função do IAM para realizar ações AWS, você é considerado principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O FAS usa as permissões do diretor chamando um AWS service (Serviço da AWS), combinadas com a solicitação AWS service (Serviço da AWS) para fazer solicitações aos serviços posteriores. As solicitações do FAS são feitas somente quando um serviço recebe uma solicitação que requer interações com outros Serviços da AWS ou com recursos para ser concluída. 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.
- Função de Serviço: uma função de serviço é uma função 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 Criando um Perfil para Delegar Permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.
- Função vinculada ao serviço — Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode assumir o perfil de executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para funções vinculadas a serviço.

- Aplicativos em execução no Amazon EC2 — Você pode usar uma função do IAM para gerenciar credenciais temporárias para aplicativos que estão sendo executados em uma instância do EC2 e fazendo AWS CLI solicitações de API. AWS É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir uma AWS função a uma instância do EC2 e disponibilizá-la para todos os seus aplicativos, você cria um perfil de instância anexado à instância. Um perfil de instância contém a perfil e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte Usar uma função do IAM para conceder permissões a aplicativos em execução nas instâncias do Amazon EC2 no Guia do usuário do IAM.

Para aprender se deseja usar perfis do IAM, consulte Quando Criar uma Função do IAM (em Vez de um Usuário) no Guia do Usuário do IAM.

Gerenciando acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política é um objeto AWS que, quando associada a uma identidade ou recurso, define suas permissões. AWS avalia essas políticas quando um principal (usuário, usuário raiz ou sessão de função) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida

ou negada. A maioria das políticas é armazenada 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 políticas AWS JSON para especificar quem tem acesso ao 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 às funções e os usuários podem assumir as funções.

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 função da AWS Management Console AWS CLI, da ou da AWS API.

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 do IAM, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e funções podem realizar, em quais recursos e em quais 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 também podem ser categorizadas como políticas em linha ou políticas gerenciadas. As políticas em linha são incorporadas diretamente a um único usuário, grupo ou função. As políticas gerenciadas são políticas autônomas que você pode associar a vários usuários, grupos e funções em seu Conta da AWS. As políticas AWS gerenciadas incluem políticas gerenciadas e políticas gerenciadas pelo cliente. Para saber como selecionar entre uma política gerenciada ou uma política em linha, consulte Selecionar entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de função do IAM e as políticas do bucket do Amazon S3. Em serviços que suportem 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. Os diretores podem incluir contas, usuários, funções, usuários federados ou. Serviços da AWS

Políticas baseadas em atributos são políticas em linha que estão localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas 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 permissão para acessar um recurso. As ACLs são semelhantes as políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

O Amazon S3 e o Amazon VPC são exemplos de serviços que oferecem suporte a ACLs. AWS WAF Saiba mais sobre ACLs em Configurações da lista de controle de acesso (ACL) no Guia do Desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais 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 e dos seus limites de permissões. As políticas baseadas em atributo 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 Entidades 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 (OU) em. AWS Organizations AWS Organizations é um serviço para agrupar e gerenciar centralmente vários Contas da AWS que sua empresa possui. 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 nas contas dos membros, incluindo cada uma Usuário raiz da conta da AWS. Para obter mais informações sobre o Organizações e SCPs, consulte Como os SCPs Funcionam no AWS Organizations Manual do Usuário do.

- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para uma função 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 atributo. 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 AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

Como Serviços da AWS trabalhar com o IAM

Para ter uma visão de alto nível de como Serviços da AWS funciona com a maioria dos recursos do IAM, consulte AWS os serviços que funcionam com o IAM no Guia do usuário do IAM.

Para saber como usar um específico AWS service (Serviço da AWS) com o IAM, consulte a seção de segurança do Guia do usuário do serviço relevante.

Solução de problemas AWS de identidade e acesso

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com AWS um IAM.

Tópicos

- Não estou autorizado a realizar uma ação em AWS
- Não estou autorizado a realizar iam: PassRole
- Quero permitir que pessoas fora da minha Conta da AWS acessem meus AWS recursos

Não estou autorizado a realizar uma ação em AWS

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 `awes:`*`GetWidget`* fictícias.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao recurso *`my-example-widget`* usando a ação `awes:`*`GetWidget`*.

Se precisar de ajuda, entre em contato com seu AWS administrador. 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 está autorizado a executar a ação `iam:PassRole`, as suas políticas devem ser atualizadas para permitir que você passe uma função para o AWS.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazê-lo, você deve ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta utilizar o console para executar uma ação no AWS. 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 precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que pessoas fora da minha Conta da AWS acessem meus AWS recursos

Você pode criar uma função que os usuários de outras contas ou pessoas fora da sua organização possam usar para acessar seus recursos. Você pode especificar quem é confiável para assumir o

perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se é AWS compatível com esses recursos, consulteComo Serviços da AWS trabalhar com o IAM.
- Para saber como fornecer acesso aos seus recursos em todos os Contas da AWS que você possui, consulte Como fornecer acesso a um usuário do IAM em outro Conta da AWS que você possui no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como fornecer acesso Contas da AWS a terceiros no Guia do usuário do IAM.
- Saiba como conceder acesso por meio da federação de identidades consultando Concedendo 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 recursos para acesso entre contas, consulte Como os perfis do IAM diferem de políticas baseadas em recursos no Guia do usuário do IAM.

Validação de conformidade para este AWS produto ou serviço

Para saber se um AWS service (Serviço da AWS) está dentro do escopo de programas de conformidade específicos, consulte Serviços da AWS Escopo por Programa de Conformidade Serviços da AWS e escolha o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de AWS conformidade Programas AWS de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixar relatórios em AWS Artifact .

Sua responsabilidade de conformidade ao usar Serviços da AWS é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentos aplicáveis. AWS fornece os seguintes recursos para ajudar na conformidade:

- Guias de início rápido sobre segurança e conformidade — Esses guias de implantação discutem considerações arquitetônicas e fornecem etapas para a implantação de ambientes básicos AWS focados em segurança e conformidade.

- Arquitetura para segurança e conformidade com a HIPAA na Amazon Web Services — Este whitepaper descreve como as empresas podem usar AWS para criar aplicativos qualificados para a HIPAA.

> **Note**
>
> Nem todos Serviços da AWS são elegíveis para a HIPAA. Para obter mais informações, consulte Referência dos Serviços Qualificados pela HIPAA.

- AWS Recursos de https://aws.amazon.com/compliance/resources/ de conformidade — Essa coleção de pastas de trabalho e guias pode ser aplicada ao seu setor e local.
- AWS Guias de conformidade do cliente — Entenda o modelo de responsabilidade compartilhada sob a ótica da conformidade. Os guias resumem as melhores práticas de proteção Serviços da AWS e mapeiam as diretrizes para controles de segurança em várias estruturas (incluindo o Instituto Nacional de Padrões e Tecnologia (NIST), o Conselho de Padrões de Segurança do Setor de Cartões de Pagamento (PCI) e a Organização Internacional de Padronização (ISO)).
- Avaliação de recursos com regras no Guia do AWS Config desenvolvedor — O AWS Config serviço avalia o quão bem suas configurações de recursos estão em conformidade com as práticas internas, as diretrizes e os regulamentos do setor.
- AWS Security Hub— Isso AWS service (Serviço da AWS) fornece uma visão abrangente do seu estado de segurança interno AWS. O Security Hub usa controles de segurança para avaliar os atributos da AWS e verificar a conformidade com os padrões e as práticas recomendadas do setor de segurança. Para obter uma lista dos serviços com suporte e controles aceitos, consulte a Referência de controles do Security Hub.
- Amazon GuardDuty — Isso AWS service (Serviço da AWS) detecta possíveis ameaças às suas cargas de trabalho Contas da AWS, contêineres e dados monitorando seu ambiente em busca de atividades suspeitas e maliciosas. GuardDuty pode ajudá-lo a atender a vários requisitos de conformidade, como o PCI DSS, atendendo aos requisitos de detecção de intrusões exigidos por determinadas estruturas de conformidade.
- AWS Audit Manager— Isso AWS service (Serviço da AWS) ajuda você a auditar continuamente seu AWS uso para simplificar a forma como você gerencia o risco e a conformidade com as regulamentações e os padrões do setor.

Esse AWS produto ou serviço segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço

e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Resiliência para este AWS produto ou serviço

A infraestrutura AWS global é construída em torno Regiões da AWS de zonas de disponibilidade.

Regiões da AWS fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância.

Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de datacenter tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte Infraestrutura AWS global.

Esse AWS produto ou serviço segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Segurança da infraestrutura para este AWS produto ou serviço

Esse AWS produto ou serviço usa serviços gerenciados e, portanto, é protegido pela segurança de rede AWS global. Para obter informações sobre serviços AWS de segurança e como AWS proteger a infraestrutura, consulte AWS Cloud Security. Para projetar seu AWS ambiente usando as melhores práticas de segurança de infraestrutura, consulte Proteção de infraestrutura no Security Pillar AWS Well-Architected Framework.

Você usa chamadas de API AWS publicadas para acessar este AWS Produto ou Serviço pela rede. Os clientes devem ser compatíveis com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Conjuntos de criptografia com Perfect Forward Secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, suporta 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 é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Esse AWS produto ou serviço segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Aplicar uma versão mínima do TLS

Para aumentar a segurança ao se comunicar com AWS os serviços, configure o AWS SDK for JavaScript para usar o TLS 1.2 ou posterior.

> **Important**
>
> A AWS SDK for JavaScript v3 negocia automaticamente a versão TLS de nível mais alto suportada por um determinado AWS endpoint de serviço. Opcionalmente, você pode aplicar uma versão mínima do TLS exigida pelo seu aplicativo, como o TLS 1.2 ou 1.3, mas observe que o TLS 1.3 não é suportado por alguns endpoints de AWS serviço, portanto, algumas chamadas podem falhar se você aplicar o TLS 1.3.

O Transport Layer Security (TLS) é um protocolo usado por navegadores da web e outros aplicativos para garantir a privacidade e a integridade dos dados trocados por meio de uma rede.

Verificar e impor o TLS no Node.js

Quando você usa o AWS SDK for JavaScript com o Node.js, a camada de segurança subjacente do Node.js é usada para definir a versão do TLS.

O Node.js 12.0.0 e posterior usam uma versão mínima do OpenSSL 1.1.1b, que oferece suporte ao TLS 1.3. A AWS SDK for JavaScript v3 usa como padrão o TLS 1.3 quando disponível, mas usa como padrão uma versão inferior, se necessário.

Verificar a versão do OpenSSL e do TLS

Para obter a versão do OpenSSL usada pelo Node.js no seu computador, execute o comando a seguir.

```
node -p process.versions
```

A versão do OpenSSL na lista é a versão usada pelo Node.js, como mostrado no exemplo a seguir.

```
openssl: '1.1.1b'
```

Para obter a versão do TLS usada pelo Node.js no seu computador, inicie o shell do Node e execute os comandos a seguir, na ordem.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

O último comando gera a versão do TLS, como mostrado no exemplo a seguir.

```
'TLSv1.3'
```

O padrão do Node.js é usar essa versão do TLS e tentará negociar outra versão do TLS se uma chamada não for bem-sucedida.

Impor uma versão mínima do TLS

O Node.js negocia uma versão do TLS quando uma chamada falha. Você pode aplicar a versão mínima permitida do TLS durante essa negociação, seja ao executar um script na linha de comando ou por solicitação em seu código. JavaScript

Para especificar a versão mínima do TLS a partir da linha de comando, você deve usar o Node.js versão 11.0.0 ou posterior. Para instalar uma versão específica do Node.js, primeiro instale o Gerenciador de versão do Node (nvm) usando as etapas encontradas em Instalação e atualização do Gerenciador de versão do Node. Execute os comandos a seguir para instalar e usar uma versão específica do Node.js.

```
nvm install 11
```

```
nvm use 11
```

Enforce TLS 1.2

Para impor que o TLS 1.2 seja a versão mínima permitida, especifique o argumento `--tls-min-v1.2` ao executar o script, como mostrado no exemplo a seguir.

```
node --tls-min-v1.2 yourScript.js
```

Para especificar a versão mínima permitida do TLS para uma solicitação específica em seu JavaScript código, use o `httpOptions` parâmetro para especificar o protocolo, conforme mostrado no exemplo a seguir.

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_2_method'
            }
        )
    })
});
```

Enforce TLS 1.3

Para impor que o TLS 1.3 seja a versão mínima permitida, especifique o argumento `--tls-min-v1.3` ao executar o script, como mostrado no exemplo a seguir.

```
node --tls-min-v1.3 yourScript.js
```

Para especificar a versão mínima permitida do TLS para uma solicitação específica em seu JavaScript código, use o `httpOptions` parâmetro para especificar o protocolo, conforme mostrado no exemplo a seguir.

```
import https from "https";
```

```
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_3_method'
            }
        )
    })
});
```

Verificar e impor o TLS em um script de navegador

Quando você usa o SDK JavaScript em um script de navegador, as configurações do navegador controlam a versão do TLS usada. A versão do TLS usada pelo navegador não pode ser descoberta nem definida por script e deve ser configurada pelo usuário. Para verificar e impor a versão do TLS usada em um script de navegador, consulte as instruções para seu navegador específico.

Microsoft Internet Explorer

1. Abra o Internet Explorer.
2. Na barra de menu, escolha a guia Ferramentas - Opções da Internet - Avançado.
3. Role para baixo até a categoria Segurança e marque manualmente a caixa de opção Usar TLS 1.2.
4. Clique em OK.
5. Feche o navegador e reinicie o Internet Explorer.

Microsoft Edge

1. Na caixa de pesquisa do menu do Windows, digite *Opções da Internet*.
2. Em Melhor correspondência, clique em Opções da Internet.
3. Na janela Propriedades da Internet, na guia Avançado, role para baixo até a seção Segurança.
4. Marque a caixa de seleção Usar TLS 1.2.

5. Clique em OK.

Google Chrome

1. Abra o Google Chrome.
2. Clique em Alt F e selecione Configurações.
3. Role para baixo e selecione Mostrar configurações avançadas....
4. Role para baixo até a seção Sistema e clique em Abrir configurações de proxy....
5. Selecione a guia Avançado.
6. Role para baixo até a categoria Segurança e marque manualmente a caixa de opção Usar TLS 1.2.
7. Clique em OK.
8. Feche seu navegador e reinicie o Google Chrome.

Mozilla Firefox

1. Abra o Firefox.
2. Na barra de endereço, digite about:config e pressione Enter.
3. No campo Pesquisar, digite tls. Localize e clique duas vezes na entrada de security.tls.version.min.
4. Defina o valor inteiro como 3 para forçar o protocolo TLS 1.2 a ser o padrão.
5. Clique em OK.
6. Feche seu navegador e reinicie o Mozilla Firefox.

Apple Safari

Não há opções para ativar os protocolos SSL. Se você estiver usando o Safari versão 7 ou superior, o TLS 1.2 será ativado automaticamente.

Migre da versão 2.x para a 3.x do AWS SDK for JavaScript

A AWS SDK for JavaScript versão 3 é uma grande reescrita da versão 2. A seção descreve as diferenças entre as duas versões e explica como migrar da versão 2 para a versão 3 do SDK para. JavaScript

Migre seu código para o SDK for JavaScript v3 usando codemod

AWS SDK for JavaScript a versão 3 (v3) vem com interfaces modernizadas para configurações e utilitários de clientes, que incluem credenciais, upload de várias partes do Amazon S3, cliente de documentos do DynamoDB, garçons e muito mais. Você pode descobrir o que mudou na v2 e nos equivalentes da v3 para cada alteração no guia de migração no repositório. AWS SDK for JavaScript GitHub

Para aproveitar ao máximo a AWS SDK for JavaScript v3, recomendamos usar os scripts de codemod descritos abaixo.

Use o codemod para migrar o código v2 existente

A coleção de scripts de codemod em aws-sdk-js-codemod ajuda a migrar seu aplicativo existente (v2) para usar APIs v3. AWS SDK for JavaScript Você pode executar a transformação da seguinte maneira.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

Por exemplo, considere que você tem o código a seguir, que cria um cliente Amazon DynamoDB a partir da v2 e chama a operação `listTables`.

```
// example.ts
import AWS from "aws-sdk";

const region = "us-west-2";
const client = new AWS.DynamoDB({ region });
client.listTables({}, (err, data) => {
  if (err)
    console.log(err, err.stack);
  else
    console.log(data);
```

```
});
```

Você pode executar a transformação `v2-to-v3` em `example.ts` da seguinte maneira.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

A transformação converterá a importação do DynamoDB em v3, criará o cliente v3 e chamará a operação `listTables` da seguinte forma.

```
// example.ts
import { DynamoDB } from "@aws-sdk/client-dynamodb";

const region = "us-west-2";
const client = new DynamoDB({ region });
client.listTables({}, (err, data) => {
  if (err)
    console.log(err, err.stack);
  else
    console.log(data);
});
```

Implementamos transformações para casos de uso comuns. Se seu código não for transformado corretamente, crie um relatório de bugs ou uma solicitação de atributo com exemplo de código de entrada e código de saída observado/esperado. Se seu caso de uso específico já foi relatado em um problema existente, mostre seu apoio por meio de um voto positivo.

Novidades da versão 3

A versão 3 do SDK para JavaScript (v3) contém os seguintes novos recursos.

Pacotes modularizados

Agora, os usuários podem usar um pacote separado para cada serviço.

Nova pilha de middleware

Agora, os usuários podem usar uma pilha de middleware para controlar o ciclo de vida de uma chamada de operação.

Além disso, o SDK é incorporado TypeScript, o que tem muitas vantagens, como digitação estática.

> **Important**
>
> Os exemplos de código para a v3 neste guia estão escritos em ECMAScript 6 (ES6). O ES6 apresenta nova sintaxe e novos atributos para tornar seu código mais moderno e legível, e muito mais. O ES6 requer que você use o Node.js versão 13.x ou superior. Para baixar e instalar a versão mais recente do Node.js, consulte Downloads do Node.js. Para ter mais informações, consulte Sintaxe ES6/CommonJS de JavaScript.

Pacotes modularizados

A versão 2 do SDK para JavaScript (v2) exigia que você usasse o AWS SDK inteiro, da seguinte forma.

```
var AWS = require("aws-sdk");
```

Carregar o SDK inteiro não é um problema se seu aplicativo estiver usando muitos AWS serviços. No entanto, se você precisar usar apenas alguns AWS serviços, isso significa aumentar o tamanho do seu aplicativo com código que você não precisa nem usa.

Na v3, você pode carregar e usar somente os AWS serviços individuais de que precisa. Isso é mostrado no exemplo a seguir, que fornece acesso ao Amazon DynamoDB (DynamoDB).

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";
```

Você não só pode carregar e usar AWS serviços individuais, mas também pode carregar e usar somente os comandos de serviço necessários. Isso é mostrado nos exemplos a seguir, que fornecem acesso ao cliente do DynamoDB e ao comando `ListTablesCommand`.

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
```

> **Important**
>
> Você não deve importar submódulos em módulos. Por exemplo, o código a seguir poderá retornar erros.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/
CognitoIdentity";
```

A seguir está o código correto.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
```

Comparação de tamanho de código

Na versão 2 (v2), um exemplo de código simples que lista todas as suas tabelas do Amazon DynamoDB `us-west-2` na região pode ter a seguinte aparência.

```
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Tables names are ", data.TableNames);
  }
});
```

A v3 tem a seguinte aparência.

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";

(async function () {
  const dbclient = new DynamoDBClient({ region: 'us-west-2' });

  try {
    const results = await dbclient.send(new ListTablesCommand);
```

```
    results.TableNames.forEach(function (item, index) {
      console.log(item);
    });
  } catch (err) {
    console.error(err)
  }
})();
```

O pacote `aws-sdk` adiciona cerca de 40 MB ao seu aplicativo. Substituir `var AWS = require("aws-sdk")` por `import {DynamoDB} from "@aws-sdk/client-dynamodb"` reduz essa sobrecarga para cerca de 3 MB. Restringir a importação apenas ao cliente do DynamoDB e ao comando `ListTablesCommand` reduz a sobrecarga para menos de 100 KB.

```
// Load the DynamoDB client and ListTablesCommand command for Node.js
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbclient = new DynamoDBClient({});
```

Chamando comandos na v3

Você pode realizar operações na v3 usando os comandos v2 ou v3. Para usar os comandos v3, você importa os comandos e os clientes do pacote de AWS serviços necessários e executa o comando usando o `.send` método usando o padrão async/await.

Para usar os comandos v2, você importa os pacotes de AWS serviços necessários e executa o comando v2 diretamente no pacote usando um padrão de retorno de chamada ou async/await.

Usando comandos v3

A v3 fornece um conjunto de comandos para cada pacote AWS de serviços para permitir que você execute operações para esse AWS serviço. Depois de instalar um Serviço da AWS , você pode navegar pelos comandos disponíveis na `node-modules/@aws-sdk/client-`*`PACKAGE_NAME`*`/commands folder.` de seu projeto.

Você deve importar os comandos que deseja usar. Por exemplo, o código a seguir carrega o serviço do DynamoDB e o comando `CreateTableCommand`.

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

Para chamar esses comandos no padrão async/await recomendado, use a seguinte sintaxe.

```
CLIENT.send(new XXXCommand);
```

O exemplo a seguir cria uma tabela do DynamoDB usando o padrão async/await recomendado.

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
const dynamodb = new DynamoDB({ region: 'us-west-2' });
var tableParams = {
  Table: TABLE_NAME
};
(async function () => {
  try {
    const data = await dynamodb.send(new CreateTableCommand(tableParams));
    console.log("Success", data);
  }
  catch (err) {
    console.log("Error", err);
  }
}) ();
```

Usando comandos v2

Para usar os comandos v2 no SDK para JavaScript, você importa os pacotes de AWS serviços completos, conforme demonstrado no código a seguir.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

Para chamar os comandos v2 no padrão async/await recomendado, use a sintaxe a seguir.

```
client.command(parameters);
```

O exemplo a seguir usa o `createTable` comando v2 para criar uma tabela do DynamoDB usando o padrão async/await recomendado.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
const dynamoDB = new DynamoDB({ region: 'us-west-2' });
var tableParams = {
  TableName: TABLE_NAME
};
```

```
async function run() => {
  try {
    const data = await dynamoDB.createTable(tableParams);
    console.log("Success", data);
  }
  catch (err) {
    console.log("Error", err);
  }
};
run();
```

O exemplo a seguir usa o `createBucket` comando v2 para criar um bucket do Amazon S3 usando o padrão de retorno de chamada.

```
const { S3 } = require('@aws-sdk/client-s3');
const s3 = new S3({ region: 'us-west-2' });
var bucketParams = {
  Bucket : BUCKET_NAME
};
function run() {
  s3.createBucket(bucketParams, function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.Location);
    }
  })
};
run();
```

Nova pilha de middleware

A v2 do SDK permitiu que você modificasse uma solicitação em vários estágios de seu ciclo de vida, anexando ouvintes de eventos à solicitação. Essa abordagem pode dificultar a depuração do que deu errado durante o ciclo de vida de uma solicitação.

Na v3, você pode usar uma nova pilha de middleware para controlar o ciclo de vida de uma chamada de operação. Essa abordagem oferece alguns benefícios. Cada estágio de middleware na pilha chama o próximo estágio de middleware depois de fazer qualquer alteração no objeto de solicitação. Isso também facilita muito a depuração de problemas na pilha, porque você pode ver exatamente quais estágios de middleware foram chamados antes do erro.

O exemplo a seguir adiciona um cabeçalho personalizado a um cliente do Amazon DynamoDB (que criamos e mostramos anteriormente) usando middleware. O primeiro argumento é uma função que aceita `next`, que é o próximo estágio de middleware na pilha a ser chamada, e `context`, que é um objeto que contém algumas informações sobre a operação que está sendo chamada. A função retorna uma função que aceita `args`, que é um objeto que contém os parâmetros passados para a operação e a solicitação. Ela retorna o resultado da chamada do próximo middleware com `args`.

```
dbclient.middlewareStack.add(
  (next, context) => args => {
    args.request.headers["Custom-Header"] = "value";
    return next(args);
  },
  {
    step: "build"
  }
);

dbclient.send(new PutObjectCommand(params));
```

O que há de diferente entre a AWS SDK for JavaScript v2 e a v3

Esta seção captura mudanças notáveis da AWS SDK for JavaScript v2 para a v3. Como a v3 é uma reescrita modular da v2, alguns conceitos básicos são diferentes entre a v2 e a v3. Você pode aprender sobre essas mudanças nas postagens do nosso blog. As seguintes postagens do blog ajudarão você a se atualizar:

- Pacotes modulares em AWS SDK for JavaScript
- Apresentando o Middleware Stack em Modular AWS SDK for JavaScript

O resumo das alterações da interface de AWS SDK for JavaScript v2 para v3 é fornecido abaixo. O objetivo é ajudá-lo a encontrar facilmente os equivalentes v3 das APIs v2 com as quais você já está familiarizado.

Construtores de clientes

Essa lista é indexada pelos parâmetros de configuração da v2.

- `computeChecksums`

  - v2: Se as somas de verificação MD5 devem ser calculadas para corpos de carga quando o serviço as aceita (atualmente suportado somente no S3).
  - v3: os comandos aplicáveis do S3 (PutObject, PutBucketCors, etc.) calcularão automaticamente as somas de verificação MD5 da carga útil da solicitação. Você também pode especificar um algoritmo de soma de verificação diferente no `ChecksumAlgorithm` parâmetro dos comandos para usar um algoritmo de soma de verificação diferente. Você pode encontrar mais informações no anúncio dos recursos do S3
- `convertResponseTypes`
  - v2: Se os tipos são convertidos ao analisar dados de resposta.
  - v3: Obsoleto. Essa opção é considerada insegura porque não converte tipos como timestamp ou binários base64 da resposta JSON.
- `correctClockSkew`
  - v2: Se deve aplicar uma correção de distorção do relógio e repetir as solicitações que falham devido a um relógio distorcido do cliente.
  - v3: Obsoleto. O SDK sempre aplica uma correção de inclinação do relógio.
- `systemClockOffset`
  - v2: um valor de deslocamento em milissegundos a ser aplicado a todos os horários de assinatura.
  - v3: Sem alteração.
- `credentials`
  - v2: As AWS credenciais com as quais assinar solicitações.
  - v3: Sem alteração. Também pode ser uma função assíncrona que retorna credenciais. Se a função retornar um`expiration (Date)`, a função será chamada novamente quando a data e hora de expiração se aproximar. Consulte a referência da v3 para obter AwsAuthInputConfig as credenciais.
- `endpointCacheSize`
  - v2: O tamanho do cache global que armazena os endpoints das operações de descoberta de endpoints.
  - v3: Sem alteração.
- `endpointDiscoveryEnabled`
  - v2: se as operações devem ser chamadas com endpoints fornecidos pelo serviço de forma dinâmica.
  - v3: Sem alteração.
- `hostPrefixEnabled`
  - v2: Se os parâmetros da solicitação devem ser agrupados com o prefixo do nome do host.

  - v3: Obsoleto. O SDK sempre injeta o prefixo do nome do host quando necessário.
- `httpOptions`

  Um conjunto de opções para passar para a solicitação HTTP de baixo nível. Essas opções são agregadas de forma diferente na v3. Você pode configurá-los fornecendo um novo`requestHandler`. Aqui está o exemplo de configuração de opções http no tempo de execução do Node.js. Você pode encontrar mais na referência da v3 para NodeHttpHandler.

  Todas as solicitações v3 usam HTTPS por padrão. Você só precisa fornecer um HttpsAgent personalizado.

```
const { Agent } = require("https");
const { Agent: HttpAgent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpsAgent: new Agent({
            /*params*/
        }),
        connectionTimeout: /*number in milliseconds*/,
        socketTimeout: /*number in milliseconds*/
    }),
});
```

  Se você estiver passando um endpoint personalizado que usa http, precisará fornecer HTTPAgent.

```
const { Agent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");

const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: new Agent({
            /*params*/
        }),
    }),
    endpoint: "http://example.com",
});
```

  Se o cliente estiver sendo executado em navegadores, um conjunto diferente de opções estará disponível. Você pode encontrar mais na referência da v3 para FetchHttpHandler.

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
const dynamodbClient = new DynamoDBClient({
    requestHandler: new FetchHttpHandler({
        requestTimeout: /* number in milliseconds */
    }),
});
```

Cada opção de `httpOptions` é especificada abaixo:

- `proxy`
  - v2: O URL por meio do qual as solicitações de proxy
  - v3: Você pode configurar um proxy com um agente seguindo Como configurar proxies para Node.js
- `agent`
  - v2: O objeto Agent com o qual realizar solicitações HTTP. Usado para agrupamento de conexões.
  - v3: Você pode configurar `httpAgent` ou `httpsAgent` conforme mostrado nos exemplos acima.
- `connectTimeout`
  - v2: define o soquete para tempo limite após não conseguir estabelecer uma conexão com o servidor após milissegundos de connectTimeout.
  - v3: `connectionTimeout` está disponível em `NodeHttpHandler` opções.
- `timeout`
  - v2: o número de milissegundos que uma solicitação pode levar antes de ser encerrada automaticamente.
  - v3: `socketTimeout` está disponível em `NodeHttpHandler` opções.
- `xhrAsync`
  - v2: se o SDK enviará solicitações HTTP assíncronas.
  - v3: Obsoleto. As solicitações são sempre assíncronas.
- `xhrWithCredentials`
  - v2: define a propriedade “withCredentials” de um objeto XML. HttpRequest
  - v3: Não disponível. O SDK herda as configurações de busca padrão
- `logger`
  - v2: um objeto que responde a .write () (como um stream) ou .log () (como o objeto de console) para registrar informações sobre solicitações.
  - v3: Sem alteração. Registros mais granulares estão disponíveis na v3.

- `maxRedirects`
  - v2: a quantidade máxima de redirecionamentos a serem seguidos para uma solicitação de serviço.
  - v3: Obsoleto. O SDK não segue redirecionamentos para evitar solicitações não intencionais entre regiões.
- `maxRetries`
  - v2: a quantidade máxima de tentativas a serem executadas para uma solicitação de serviço.
  - v3: Alterado para. `maxAttempts` Veja mais na referência da v3 para RetryInputConfig. Observe que `maxAttempt` deveria ser`maxRetries + 1`.
- `paramValidation`
  - v2: Se os parâmetros de entrada devem ser validados em relação à descrição da operação antes de enviar a solicitação.
  - v3: Obsoleto. O SDK não faz validação no lado do cliente em tempo de execução.
- `region`
  - v2: A região para a qual enviar solicitações de serviço.
  - v3: Sem alteração. Também pode ser uma função assíncrona que retorna uma string de região.
- `retryDelayOptions`
  - v2: Um conjunto de opções para configurar o atraso de repetição em erros que podem ser repetidos.
  - v3: Obsoleto. O SDK oferece suporte a uma estratégia de repetição mais flexível com a opção de construtor `retryStrategy` do cliente. Veja mais na referência da v3
- `s3BucketEndpoint`
  - v2: se o endpoint fornecido endereça um bucket individual (falso se endereçar ao endpoint raiz da API).
  - v3: Alterado para. `bucketEndpoint` Veja mais na referência v3 para BucketEndpoint. Observe que, quando definido como`true`, você especifica o endpoint da solicitação no parâmetro da `Bucket` solicitação, o endpoint original será sobrescrito. Já na v2, o endpoint da solicitação no construtor do cliente sobrescreve o parâmetro da solicitação. `Bucket`
- `s3DisableBodySigning`
  - v2: Se a assinatura corporal do S3 deve ser desativada ao usar a versão v4 da assinatura.
  - v3: Renomeado para `applyChecksum`
- `s3ForcePathStyle`
  - v2: se os URLs de estilo de caminho devem ser forçados para objetos do S3.
  - v3: Renomeado para `forcePathStyle`
- `s3UseArnRegion`

  - v2: se a região solicitada deve ser substituída pela região inferida do ARN do recurso solicitado.
  - v3: Renomeado para `useArnRegion`
- `s3UsEast1RegionalEndpoint`
  - v2: Quando a região está definida como 'us-east-1', seja para enviar a solicitação s3 para endpoints globais ou endpoints regionais 'us-east-1 '.
  - v3: Obsoleto. O cliente S3 sempre usará o endpoint regional se a região estiver definida como. `us-east-1` Você pode definir a região para enviar solicitações `aws-global` para o endpoint global do S3.
- `signatureCache`
  - v2: se a assinatura com a qual assinar solicitações (substituindo a configuração da API) está armazenada em cache.
  - v3: Obsoleto. O SDK sempre armazena em cache as chaves de assinatura com hash.
- `signatureVersion`
  - v2: a versão da assinatura com a qual assinar solicitações (substituindo a configuração da API).
  - v3: Obsoleto. O Signature V2 suportado no SDK v2 foi descontinuado por AWS. A v3 suporta apenas a assinatura v4.
- `sslEnabled`
  - v2: Se o SSL está habilitado para solicitações.
  - v3: Renomeado para. `tls`
- `stsRegionalEndpoints`
  - v2: Se deve enviar uma solicitação sts para endpoints globais ou endpoints regionais.
  - v3: Obsoleto. O cliente STS sempre usará endpoints regionais se definido para uma região específica. Você pode definir a região para enviar `aws-global` a solicitação ao endpoint global STS.
- `useAccelerateEndpoint`
  - v2: se deve usar o endpoint Accelerate com o serviço S3.
  - v3: Sem alteração.

Provedores de credenciais

Na v2, o SDK para JavaScript fornece uma lista de provedores de credenciais para escolher, bem como uma cadeia de fornecedores de credenciais, disponível por padrão no Node.js, que tenta carregar AWS as credenciais de todos os provedores mais comuns. O SDK para JavaScript v3 simplifica a interface do provedor de credenciais, facilitando o uso e a criação de provedores de credenciais personalizados. Além de uma nova cadeia de fornecedores de credenciais, o SDK

para JavaScript v3 fornece uma lista de provedores de credenciais com o objetivo de fornecer o equivalente à v2.

Aqui estão todos os provedores de credenciais na v2 e seus equivalentes na v3.

Provedor de credenciais padrão

O provedor de credenciais padrão é como o SDK JavaScript resolve a AWS credencial se você não fornecer uma explicitamente.

- v2: A CredentialProvidercadeia em Node.js resolve a credencial das fontes na seguinte ordem:
  - Variável ambiental
  - Arquivo de credenciais compartilhado
  - Credenciais de contêiner ECS
  - Processo externo de desova
  - Token OIDC do arquivo especificado
  - Metadados da instância EC2

  Se um dos provedores de credenciais acima não conseguir resolver a AWS credencial, a cadeia voltará para o próximo provedor até que uma credencial válida seja resolvida, e a cadeia gerará um erro quando todos os provedores falharem.

  Nos tempos de execução do Browser e do React Native, a cadeia de credenciais está vazia e as credenciais devem ser definidas explicitamente.
- v3: DefaultProvider. As fontes de credenciais e a ordem de fallback não mudam na v3. Ele também oferece suporte a AWS IAM Identity Center credenciais.

Credenciais temporárias

- v2: `ChainableTemporaryCredentials`representa credenciais temporárias recuperadas de. `AWS.STS` Sem nenhum parâmetro extra, as credenciais serão obtidas da `AWS.STS.getSessionToken()` operação. Se uma função do IAM for fornecida, a `AWS.STS.assumeRole()` operação será usada para buscar credenciais para a função. `AWS.ChainableTemporaryCredentials`difere da `AWS.TemporaryCredentials` forma como as MasterCredentials e as atualizações são tratadas. `AWS.ChainableTemporaryCredentials`atualiza as credenciais expiradas usando as MasterCredentials passadas pelo usuário para suportar o encadeamento de credenciais STS. No entanto, reduz `AWS.TemporaryCredentials` recursivamente as MasterCredentials

durante a instanciação, impedindo a capacidade de atualizar credenciais que exigem credenciais intermediárias temporárias.

O original `TemporaryCredentials`foi descontinuado em favor do `ChainableTemporaryCredentials` v2.

- v3: `fromTemporaryCredentials`. Você pode ligar `fromTemporaryCredentials()` do `@aws-sdk/credential-providers` pacote. Veja um exemplo abaixo:

```
import { FooClient } from "@aws-sdk/client-foo";
import { fromTemporaryCredentials } from "@aws-sdk/credential-providers"; // ES6
 import
// const { FooClient } = require("@aws-sdk/client-foo");
// const { fromTemporaryCredentials } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const sourceCredentials = {
  // A credential can be a credential object or an async function that returns a
 credential object
};
const client = new FooClient({
  credentials: fromTemporaryCredentials({
    masterCredentials: sourceCredentials,
    params: { RoleArn },
  }),
});
```

Credenciais de identidade do Amazon Cognito

Carregue credenciais do serviço Amazon Cognito Identity, normalmente usado em navegadores.

- v2: `CognitoIdentityCredentials`representa as credenciais recuperadas da STS Web Identity Federation usando o serviço Amazon Cognito Identity.
- v3: `Cognito Identity Credential Provider`O `@aws/credential-providers`pacote fornece duas funções de provedor de credenciais, uma das quais `fromCognitoIdentity`recebe uma ID de identidade e chamadas`cognitoIdentity:GetCredentialsForIdentity`, enquanto a outra `fromCognitoIdentityPool`recebe uma ID de grupo de identidades, chama a primeira invocação e, `cognitoIdentity:GetId` em seguida, chama. `fromCognitoIdentity` As invocações subsequentes deste último não são invocadas novamente. GetId

O provedor implementa o “Fluxo simplificado” descrito no Guia do Desenvolvedor do Amazon Cognito. O “Classic Flow”, que envolve chamadas `cognito:GetOpenIdToken` e depois chamadas, não *`sts:AssumeRoleWithWebIdentity`* é suportado. Abra uma solicitação de recurso para nós, se precisar.

```
// fromCognitoIdentityPool example
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers"; // ES6
 import
// const { fromCognitoIdentityPool } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

```
// fromCognitoIdentity example
import { fromCognitoIdentity } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromCognitoIdentity } = require("@aws-sdk/credential-provider-cognito-
identity"); // CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentity({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityId: "us-east-1:128d0a74-c82f-4553-916d-90053e4a8b0f",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
```

```
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

Credencial de metadados do EC2 (IMDS)

Representa as credenciais recebidas do serviço de metadados em uma instância do Amazon EC2.

- v2: **`EC2MetadataCredentials`**
- v3: `fromInstanceMetadata`: Cria um provedor de credenciais que fornecerá credenciais do Amazon EC2 Instance Metadata Service.

```
import { fromInstanceMetadata } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromInstanceMetadata } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const client = new FooClient({
  credentials: fromInstanceMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

Credenciais ECS

Representa as credenciais recebidas do URL especificado. Esse provedor solicitará credenciais temporárias do URI especificado pela `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` ou pela variável de `AWS_CONTAINER_CREDENTIALS_FULL_URI` ambiente.

- v2: `ECSCredentials` ou `RemoteCredentials`.
- v3: `fromContainerMetadata`cria um provedor de credenciais que fornecerá credenciais do Amazon ECS Container Metadata Service.

```
import { fromContainerMetadata } from "@aws-sdk/credential-providers"; // ES6 import

const client = new FooClient({
  credentials: fromContainerMetadata({
    maxRetries: 3, // Optional
```

```
    timeout: 0, // Optional
  }),
});
```

Credenciais do sistema de arquivos

- v2: `FileSystemCredentials`representa as credenciais de um arquivo JSON no disco.
- v3: Obsoleto. Você pode ler explicitamente o arquivo JSON e fornecer ao cliente. Abra uma solicitação de recurso para nós, se precisar.

Provedor de credenciais SAML

- v2: `SAMLCredentials`representa as credenciais recuperadas do suporte ao STS SAML.
- v3: Não disponível. Abra uma solicitação de recurso para nós, se precisar.

Credenciais do arquivo de credencial compartilhado

Carrega as credenciais do arquivo de credenciais compartilhado (usando como padrão `~/.aws/credentials` ou definido pela variável de ambiente). `AWS_SHARED_CREDENTIALS_FILE` Esse arquivo é compatível com diferentes AWS SDKs e ferramentas. Você pode consultar o documento de arquivos de configuração e credenciais compartilhados para obter mais informações.

- v2: **`SharedIniFileCredentials`**
- v3: `fromIni`.

```
import { fromIni } from "@aws-sdk/credential-providers";
// const { fromIni } from("@aws-sdk/credential-providers");

const client = new FooClient({
  credentials: fromIni({
    configFilepath: "~/.aws/config", // Optional
    filepath: "~/.aws/credentials", // Optional
    mfaCodeProvider: async (mfaSerial) => {
      // implement a pop-up asking for MFA code
      return "some_code";
    }, // Optional
    profile: "default", // Optional
    clientConfig: { region }, // Optional
```

```
  }),
});
```

Credenciais de identidade na Web

Recupera as credenciais usando o token OIDC de um arquivo no disco. É comumente usado no EKS.

- v2: `TokenFileWebIdentityCredentials`.
- v3: **`fromTokenFile`**

```
import { fromTokenFile } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromTokenFile } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromTokenFile({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Credenciais da Web Identity Federation

Recupera credenciais do suporte da federação de identidade web STS.

- v2: **`WebIdentityCredentials`**
- v3: **`fromWebToken`**

```
import { fromWebToken } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromWebToken } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromWebToken({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
```

```
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Considerações sobre o Amazon S3

Upload de várias partes do Amazon S3

Na v2, o cliente Amazon S3 contém `upload()`uma operação que suporta o upload de objetos grandes com o recurso de upload de várias partes oferecido pelo Amazon S3.

Na v3, o `@aws-sdk/lib-storage`pacote está disponível. Ele suporta todos os recursos oferecidos na `upload()` operação v2 e suporta tanto o Node.js quanto o tempo de execução dos navegadores.

URL pré-assinada do Amazon S3

Na v2, o cliente Amazon S3 contém `getSignedUrl()`as operações `getSignedUrlPromise()`e para gerar uma URL que os usuários podem usar para carregar ou baixar objetos do Amazon S3.

Na v3, o `@aws-sdk/s3-request-presigner`pacote está disponível. Este pacote contém as funções para ambas `getSignedUrl()` as `getSignedUrlPromise()` operações. Esta postagem do blog discute os detalhes desse pacote.

Redirecionamentos de região do Amazon S3

Se uma região incorreta for passada para o cliente Amazon S3 e um erro subsequente `PermanentRedirect` (status 301) for gerado, o cliente Amazon S3 na v3 suportará redirecionamentos de região (anteriormente conhecido como Amazon S3 Global Client na v2). Você pode usar a `followRegionRedirects`bandeira na configuração do cliente para fazer com que o cliente Amazon S3 siga os redirecionamentos da região e apoie sua função como cliente global.

> **Note**
>
> Observe que esse recurso pode resultar em latência adicional, pois as solicitações com falha são repetidas com uma região corrigida ao receber um `PermanentRedirect` erro

com status 301. Esse recurso só deve ser usado se você não souber a região do (s) seu (s) bucket (s) com antecedência.

Histórico do documento para a AWS SDK for JavaScript versão 3

Histórico do documento

A tabela a seguir descreve as alterações importantes na versão V3 do AWS SDK for JavaScript, de 20 de outubro de 2020 em diante. Para receber notificações sobre atualizações dessa documentação, inscreva-se em um feed RSS.

Alteração	Descrição	Data
Comunicado	Banner superior atualizado com um end-of-support lembrete para o Internet Explorer 11.	23 de setembro de 2022
Atualizações menores	Atualizações menores para esclarecer e resolver links quebrados. Links de conscientização adicionados ao Guia de referência de AWS SDKs e ferramentas.	22 de agosto de 2022
Aplique uma versão mínima do TLS	Adição de informações sobre TLS 1.3.	31 de março de 2022
AWS Lambda Tutorial atualizado	Foi adicionado um tutorial que demonstra como criar um aplicativo baseado em navegador para enviar dados a uma tabela do Amazon DynamoDB.	20 de outubro de 2020
Definir credenciais no tópico Node.js atualizado	Atualize o tópico sobre a configuração de credenciais	20 de outubro de 2020

	no Node.js para a AWS SDK for JavaScript V3.	
Migre para a v3	Tópico adicionado para descrever como migrar para a AWS SDK for JavaScript v3.	20 de outubro de 2020
Conceitos básicos	Tópicos atualizados para começar a usar o navegador e começar a usar o Node.js para AWS SDK for JavaScript V3.	20 de outubro de 2020
Criador de navegadores	As informações sobre o AWS Browser Builder foram removidas porque não são necessárias para a AWS SDK for JavaScript V3.	20 de outubro de 2020
Exemplos de serviços do Amazon Transcribe atualizados	Exemplos atualizados do serviço Amazon Transcribe para a V3. AWS SDK for JavaScript	20 de outubro de 2020
Exemplos de serviços do Amazon Simple Notification Service atualizados	Exemplos atualizados do serviço Amazon Simple Notification Service para a AWS SDK for JavaScript V3.	20 de outubro de 2020
Exemplos de serviços do Amazon Simple Email Service atualizados	Exemplos atualizados do serviço Amazon Simple Email Service para a AWS SDK for JavaScript V3.	20 de outubro de 2020
Exemplos de serviços do Amazon Redshift atualizados	Exemplos de serviços atualizados do Amazon Redshift para AWS SDK for JavaScript a V3.	20 de outubro de 2020

Exemplos de serviços do Amazon Lex atualizados	Exemplos atualizados do serviço Amazon Lex para a AWS SDK for JavaScript V3.	20 de outubro de 2020
Exemplos de serviços do Amazon DynamoDB atualizados	Exemplos de serviços atualizados do Amazon DynamoDB para a V3. AWS SDK for JavaScript	20 de outubro de 2020
AWS Elemental MediaConvert exemplos de serviços atualizados	Exemplos AWS Elemental MediaConvert de serviços atualizados para a AWS SDK for JavaScript V3.	20 de outubro de 2020
AWS Lambda exemplos de serviços atualizados	Exemplos AWS Lambda de serviços atualizados para a AWS SDK for JavaScript V3.	20 de outubro de 2020
AWS SDK for JavaScript Prévia do Guia do Desenvolvedor V3	Versão de pré-lançamento lançada do Guia do desenvolvedor AWS SDK for JavaScript V3.	19 de outubro de 2020