



Guía del desarrollador para la versión 3 de SDK

AWS SDK for JavaScript



AWS SDK for JavaScript: Guía del desarrollador para la versión 3 de SDK

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

.....	ix
¿Cuál es el AWS SDK for JavaScript?	1
Introducción al SDK	1
Mantenimiento y compatibilidad de las versiones principales del SDK	2
Uso del SDK con Node.js	2
Uso del SDK con AWS Cloud9	2
Uso del SDK con AWS Amplify	2
Uso del SDK con navegadores web	3
Uso de navegadores en la versión 3	3
Casos de uso comunes	4
Acerca de los ejemplos	4
Recursos	4
Introducción	6
Autenticación de SDK con AWS	6
Inicie una sesión en el portal de AWS acceso	8
Información adicional de autenticación	8
Introducción a Node.js	9
El escenario	9
Requisitos previos	9
Paso 1: Configurar la estructura de paquetes e instalar los paquetes de cliente	10
Paso 2: Agregar las importaciones y el código de SDK necesarios	11
Paso 3: Ejecutar el ejemplo	13
Comenzar en el navegador	13
El escenario	14
Paso 1: Crear un grupo de identidades de Amazon Cognito y un rol de IAM	14
Paso 2: Añadir una política al rol de IAM creado	15
Paso 3: Agregar un bucket y un objeto de Amazon S3	16
Paso 4: Configurar el código del navegador	17
Paso 5: Ejecución del ejemplo	18
Limpieza	18
Configurar el SDK para JavaScript	19
Requisitos previos	19
Configure un entorno AWS Node.js	19
Navegadores web compatibles	20

Instalar el SDK	21
Cargue el SDK	22
Configurar el SDK para JavaScript	23
Configuración según el servicio	23
Establezca la configuración por servicio	24
Establezca la región AWS	24
En un constructor de clase de cliente	25
Usa una variable de entorno	25
Usa un archivo de configuración compartido	25
Orden de prioridad para establecer la región	26
Establece las credenciales	26
Prácticas recomendadas para las credenciales	26
Establecer credenciales en Node.js	27
Establecer credenciales en un navegador web	31
Consideraciones de Node.js	34
Utilice los módulos Node.js integrados	34
Utilice paquetes npm	35
Configurar MaxSockets en Node.js	35
Reutilice las conexiones con keep-alive en Node.js	36
Configure los proxies para Node.js	37
Registre paquetes de certificados en Node.js	38
Consideraciones sobre los scripts de navegador	38
Cree el SDK para navegadores	39
Uso compartido de recursos entre orígenes (CORS)	40
Paquete con webpack	44
Trabaje con AWS servicios	49
Crea y llama a objetos de servicio	49
Especifique los parámetros del objeto de servicio	50
Llame a los servicios de forma asíncrona	50
Administra las llamadas asíncronas	51
Usa async/await	52
Usa promesas	53
Utilice una función de devolución de llamada	54
Cree solicitudes de clientes de servicio	55
Gestione las respuestas de los clientes del servicio	57
Acceda a los datos devueltos en la respuesta	57

Acceder a la información sobre errores	57
Trabaja con JSON	57
Parámetros de JSON como objeto de servicio	58
Subconjunto de ejemplos de código con orientación	59
Sintaxis ES6/commonJS de JavaScript	60
Ejemplos de Amazon DynamoDB	63
Ejemplos de AWS Elemental MediaConvert	88
Ejemplos de Lambda	111
Ejemplos de Amazon Lex	111
Ejemplos de Amazon Polly	112
Ejemplos de Amazon Redshift	115
Ejemplos de Amazon SES	123
Ejemplos de Amazon SNS	153
Ejemplos de Amazon Transcribe	188
Servicio cruzado: configurar Node.js en una instancia de Amazon EC2	200
Servicio cruzado: aplicación para enviar datos	202
Servicio cruzado: aplicación de transcripción	210
Servicio cruzado: Amazon API Gateway y Lambda	222
Servicio cruzado: flujos de trabajo sin servidor con Step Functions	238
Servicio cruzado: eventos de Lambda programados	253
Servicio cruzado: ejemplo de Amazon Lex	265
Servicio cruzado: aplicación de mensajería	279
AWS Cloud9 Úselo con el SDK para JavaScript	293
Paso 1: Configura tu AWS cuenta para usar AWS Cloud9	293
Paso 2: Configura tu entorno de AWS Cloud9 desarrollo	294
Paso 3: Configura el SDK para JavaScript	294
Para configurar el SDK JavaScript de Node.js	294
Para configurar el SDK JavaScript en el navegador	295
Paso 4: Descargue el código de ejemplo	295
Paso 5: Ejecute y depure el código de ejemplo	296
Ejemplos de código	297
Acciones y escenarios	297
Auto Scaling	299
Amazon Bedrock	342
Amazon Bedrock Runtime	346
Agentes para Amazon Bedrock	390

Agentes para Amazon Bedrock Runtime	404
CloudWatch	407
CloudWatch Eventos	422
CloudWatch Registros	430
CodeBuild	448
Amazon Cognito Identity Provider	451
Amazon DocumentDB	471
DynamoDB	472
Amazon EC2	532
Elastic Load Balancing: versión 2	615
EventBridge	665
AWS Glue	672
HealthImaging	699
IAM	759
Kinesis	871
Lambda	876
Amazon Personalize	901
Eventos de Amazon Personalize	918
Versión ejecutable de Amazon Personalize	922
Amazon Pinpoint	926
Amazon Redshift	936
Amazon S3	942
S3 Glacier	1014
SageMaker	1018
Secrets Manager	1057
Amazon SES	1059
Amazon SNS	1083
Amazon SQS	1121
Step Functions	1161
AWS STS	1163
AWS Support	1166
Amazon Transcribe	1184
Ejemplos de servicios cruzados	1193
Cree una aplicación Amazon Transcribe	1194
Creación de una aplicación de streaming de Amazon Transcribe	1195
Creación de una aplicación para enviar datos a una tabla de DynamoDB	1195

Crear un chatbot de Amazon Lex	1196
Creación de una aplicación sin servidor para administrar fotos	1196
Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB ...	1197
Crear un rastreador de elementos de trabajo de Aurora Serverless	1197
Creación de una aplicación de exploración de Amazon Textract	1198
Creación de una aplicación para analizar los comentarios de los clientes	1198
Detección de EPI en imágenes	1202
Detectar objetos en imágenes	1203
Detecte personas y objetos en un video	1204
Invocación de una función de Lambda desde un navegador	1204
Uso de API Gateway para invocar una función de Lambda	1205
Usar Step Functions para invocar funciones de Lambda	1206
Usar eventos programados para invocar una función de Lambda	1206
Seguridad	1208
Protección de datos	1208
Identity and Access Management	1210
Público	1210
Autenticación con identidades	1211
Administración de acceso mediante políticas	1214
¿Cómo Servicios de AWS trabajar con IAM	1217
Solución de problemas de AWS identidad y acceso	1217
Validación de la conformidad	1219
Resiliencia	1221
Seguridad de infraestructuras	1221
Aplicar una versión mínima de TLS	1222
Verificar y aplicar TLS en Node.js	1222
Verificar y aplicar TLS en un script de navegador	1225
Migre a la versión 3	1227
Migre a la versión 3 con codemod	1227
Usa codemod para migrar el código de la versión 2 existente	1227
¿Qué novedades incluye la versión 3?	1228
Paquetes modularizados	1229
Comparación del tamaño del código	1230
Comandos de llamada en la v3	1231
Nueva pila de middleware	1233
¿Qué diferencia hay entre la v2 y la v3	1234

Constructores de clientes	1235
Proveedores de credencial	1240
Consideraciones sobre Amazon S3	1246
Historial de documentos	1248
Historial de documentos	1248

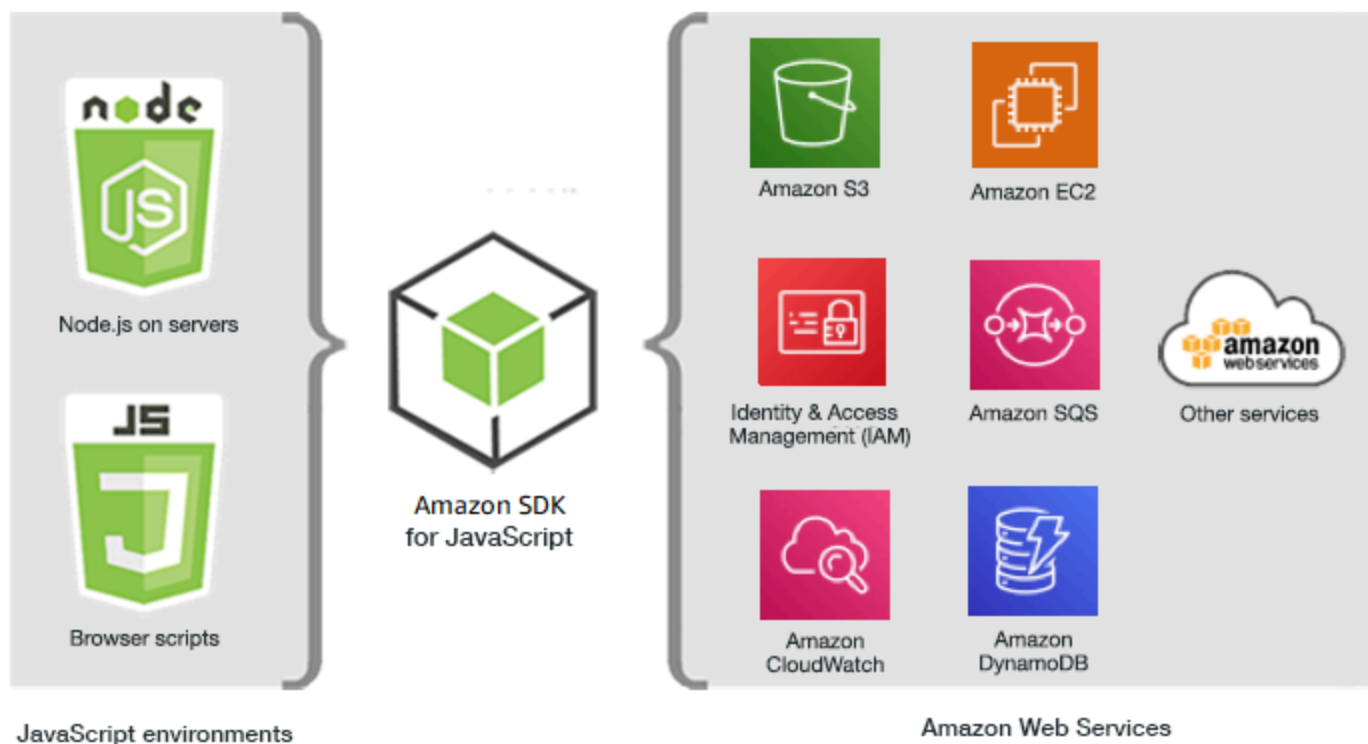
La [Guía de referencia de la API de AWS SDK for JavaScript V3](#) describe en detalle todas las operaciones de la API para la versión 3 (V3) de AWS SDK for JavaScript.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.

¿Cuál es el AWS SDK for JavaScript?

Bienvenido a la guía para AWS SDK for JavaScript desarrolladores. Esta guía proporciona información general acerca de la instalación y la configuración de AWS SDK for JavaScript. También muestra ejemplos y tutoriales sobre la ejecución de varios AWS servicios mediante el AWS SDK for JavaScript.

La [guía de referencia de la API AWS SDK for JavaScript v3](#) proporciona una JavaScript API para AWS los servicios. Puede usar la JavaScript API para crear bibliotecas o aplicaciones para [Node.js](#) o el navegador.



Introducción al SDK

Si está preparado para ponerse manos a la obra con el SDK, siga los ejemplos que se muestran en [Introducción](#).

Para configurar su entorno de desarrollo, consulte [Configurar el SDK para JavaScript](#).

Si actualmente utilizas la versión 2.x del SDK JavaScript, consulta [Migrar a la versión 3 para obtener instrucciones](#) específicas.

Si está buscando ejemplos de código para Servicios de AWS, consulte [Ejemplos de código de SDK para JavaScript \(v3\)](#)

Mantenimiento y compatibilidad de las versiones principales del SDK

Para obtener información sobre el mantenimiento y la compatibilidad con las principales versiones del SDK y sus dependencias subyacentes, consulte lo siguiente en la [Guía de Referencia de SDK y herramientas de AWS](#):

- [AWS Política de mantenimiento de SDK y herramientas](#)
- [AWS Matriz de soporte de versiones de SDK y herramientas](#)

Uso del SDK con Node.js

Node.js es un entorno de ejecución multiplataforma para ejecutar aplicaciones del lado del servidor JavaScript. Puede configurar Node.js en una instancia de Amazon Elastic Compute Cloud (Amazon EC2) para que se ejecute en un servidor. También puede usar Node.js para escribir funciones bajo demanda. AWS Lambda

El uso del SDK para Node.js difiere de la forma en que se usa JavaScript en un navegador web. La diferencia estriba en la forma en que carga el SDK y cómo obtiene las credenciales necesarias para tener acceso a servicios web específicos. Cuando el uso de determinadas API concretas varía entre Node.js y el navegador, destacamos dichas diferencias.

Uso del SDK con AWS Cloud9

También puede desarrollar aplicaciones de Node.js utilizando JavaScript el SDK del AWS Cloud9 IDE. Para obtener más información sobre su uso AWS Cloud9 con el SDK para JavaScript, consulte [AWS Cloud9 Utilízalo con AWS SDK for JavaScript](#).

Uso del SDK con AWS Amplify

Para aplicaciones web, móviles e híbridas basadas en navegador, también puedes usar la [AWS Amplify biblioteca](#) en. GitHub Amplía el SDK y proporciona una interfaz declarativa. JavaScript

Note

Es posible que los marcos como Amplify no ofrezcan la misma compatibilidad con el navegador que el SDK. JavaScript Consulte la documentación de los marcos para obtener más información.

Uso del SDK con navegadores web

Todos los principales navegadores web admiten la ejecución de JavaScript. JavaScript el código que se ejecuta en un navegador web a menudo se denomina del lado del cliente JavaScript.

Para obtener una lista de los navegadores compatibles con AWS SDK for JavaScript, consulte.

[Navegadores web compatibles](#)

El uso del SDK JavaScript en un navegador web difiere de la forma en que se utiliza para Node.js. La diferencia estriba en la forma en que carga el SDK y cómo obtiene las credenciales necesarias para tener acceso a servicios web específicos. Cuando el uso de determinadas API concretas varía entre Node.js y el navegador, destacamos dichas diferencias.

Uso de navegadores en la versión 3

La versión 3 le permite agrupar e incluir en el navegador solo el SDK para los JavaScript archivos que necesite, lo que reduce la sobrecarga.

Para utilizar la versión 3 del SDK JavaScript en sus páginas HTML, debe agrupar los módulos de cliente necesarios y todas JavaScript las funciones necesarias en un solo JavaScript archivo mediante Webpack y añadirlo en una etiqueta <head> de script en las páginas HTML. Por ejemplo:

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

Note

Para obtener más información sobre Webpack, consulte [Combine aplicaciones con webpack](#).

Para usar la versión 2 del SDK JavaScript, debe agregar una etiqueta de script que apunte a la versión más reciente del SDK de la versión 2. Para obtener más información, consulta el [ejemplo de la Guía para AWS SDK for JavaScript desarrolladores de la versión 2](#).

Casos de uso comunes

El uso del SDK JavaScript en los scripts del navegador permite realizar una serie de casos de uso convincentes. Estas son varias ideas sobre cosas que puede crear en una aplicación de navegador mediante el uso del SDK para acceder JavaScript a varios servicios web.

- Cree una consola de AWS servicios personalizada en la que pueda acceder a las funciones de todas las regiones y servicios y combínelas para satisfacer mejor las necesidades de su organización o proyecto.
- Usar Amazon Cognito para habilitar el acceso de usuarios autenticados a sus aplicaciones y sitios web de navegador, incluido el uso de la autenticación de terceros de Facebook y otros.
- Usar Amazon Kinesis para procesar flujos de clics u otros datos de marketing en tiempo real.
- Usar Amazon DynamoDB para la persistencia de datos sin servidor, como las preferencias de usuarios individuales para los visitantes de su sitio web o usuarios de la aplicación.
- AWS Lambda Utilícela para encapsular una lógica patentada que puede invocar desde los scripts del navegador sin descargar ni revelar su propiedad intelectual a los usuarios.

Acerca de los ejemplos

Puedes buscar ejemplos en el SDK en el repositorio de JavaScript ejemplos de [AWS código](#).

Recursos

Además de esta guía, están disponibles los siguientes recursos en línea sobre el SDK para JavaScript desarrolladores:

- [AWS SDK for JavaScript Guía de referencia de la API V3](#)
- [AWS Guía de referencia de los SDK y las herramientas](#): contiene la configuración, las funciones y otros conceptos fundamentales comunes a los SDK. AWS
- [JavaScript Blog para desarrolladores](#)
- [AWS JavaScript Foro](#)
- [JavaScript ejemplos en el catálogo AWS de códigos](#)
- [AWS Repositorio de ejemplos de código](#)
- [Canal de Gitter](#)
- [Desbordamiento de pila](#)

- [Preguntas de Stack Overflow etiquetadas como AWS-sdk-js](#)
- GitHub
 - [Origen del SDK](#)
 - [Origen de la documentación](#)

Comience con el AWS SDK for JavaScript

AWS SDK for JavaScript Proporciona acceso a los servicios web en un navegador o en un entorno Node.js. En esta sección se incluyen ejercicios de introducción que muestran cómo trabajar con el SDK JavaScript en cada uno de estos JavaScript entornos.

Note

Puede desarrollar aplicaciones de Node.js y, JavaScript para aplicaciones basadas en navegador, utilizar el SDK JavaScript en el AWS Cloud9 IDE. Para ver un ejemplo de cómo usarlo AWS Cloud9 para el desarrollo de Node.js, consulte. [AWS Cloud9 Utilízalo con AWS SDK for JavaScript](#)

Temas

- [Autenticación de SDK con AWS](#)
- [Introducción a Node.js](#)
- [Comenzar en el navegador](#)

Autenticación de SDK con AWS

Debes establecer cómo se autentica tu código AWS al desarrollar con Servicios de AWS. Puedes configurar el acceso programático a AWS los recursos de diferentes maneras en función del entorno y del AWS acceso del que dispongas.

Para elegir el método de autenticación y configurarlo para el SDK, consulte [Autenticación y acceso](#) en la Guía de referencia de herramientas y SDK de AWS .

Recomendamos a los nuevos usuarios que desarrollen sus proyectos de forma local y que no cuenten con un método de autenticación de su empresa para realizar la configuración AWS IAM Identity Center. Este método incluye la instalación del AWS CLI para facilitar la configuración y para iniciar sesión con regularidad en el portal de AWS acceso. Si elige este método, su entorno debería contener los siguientes elementos después de completar el procedimiento de [Autenticación del Centro de Identidad de IAM](#) descrito en la Guía de referencia de las herramientas y los SDK de AWS:

- El AWS CLI, que se utiliza para iniciar una sesión en el portal de AWS acceso antes de ejecutar la aplicación.

- Un [archivo config compartido de AWS](#) que tiene un perfil [default] con un conjunto de valores de configuración a los que se puede hacer referencia desde el SDK. Para encontrar la ubicación de este archivo, consulte [Ubicación de los archivos compartidos](#) en la Guía de referencia de las herramientas y los SDK de AWS.
- El archivo compartido de config establece la configuración de [region](#). Esto establece el valor predeterminado Región de AWS que el SDK usa para AWS las solicitudes. Esta región se usa para las solicitudes de servicio del SDK que no tienen especificadas una región.
- El SDK utiliza la [configuración de proveedor de token de SSO](#) del perfil para adquirir las credenciales antes de enviar las solicitudes a AWS. El sso_role_name valor, que es un rol de IAM conectado a un conjunto de permisos del Centro de Identidad de IAM, permite el acceso a los que Servicios de AWS se utilizan en la aplicación.

El siguiente archivo config de ejemplo muestra la configuración de un perfil predeterminado con el proveedor de token de SSO. La configuración sso_session del perfil hace referencia a la [sección llamada sso-session](#). La sso-session sección contiene la configuración para iniciar una sesión en el portal de AWS acceso.

```
[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
```

La AWS SDK for JavaScript versión 3 no necesita añadir paquetes adicionales (por ejemplo, SSO ySSO0IDC) a la aplicación para utilizar la autenticación del IAM Identity Center.

Para obtener más información sobre el uso explícito de este proveedor de credenciales, consulte [fromSSO\(\)](#) en el sitio web de npm (administrador de paquetes Node.js).

Inicie una sesión en el portal de AWS acceso

Antes de ejecutar una aplicación para acceder Servicios de AWS, necesita una sesión activa en el portal de AWS acceso para que el SDK utilice la autenticación del IAM Identity Center a fin de resolver las credenciales. En función de la duración de las sesiones configuradas, el acceso terminará por caducar y SDK detectará un error de autenticación. Para iniciar sesión en el portal de AWS acceso, ejecute el siguiente comando en. AWS CLI

```
aws sso login
```

Si sigue la guía y utiliza una configuración de perfil predeterminada, no necesita llamar al comando con una opción `--profile`. Si la configuración del proveedor de token de SSO utiliza un perfil con nombre, el comando es `aws sso login --profile named-profile`.

Si lo desea, puede comprobar si ya tiene una sesión activa, ejecute el siguiente AWS CLI comando.

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

Si su sesión está activa, la respuesta a este comando debe indicar la cuenta y el conjunto de permisos del Centro de identidades de IAM configurados en el archivo `config` compartido.

Note

Si ya tiene una sesión activa en el portal de AWS acceso y la ejecuta `aws sso login`, no tendrá que proporcionar credenciales.

Es posible que el proceso de inicio de sesión le pida que permita el AWS CLI acceso a sus datos. Como AWS CLI se basa en el SDK para Python, los mensajes de permiso pueden contener variaciones del `botocore` nombre.

Información adicional de autenticación

Los usuarios humanos, que también reciben el nombre de identidades humanas, son las personas, los administradores, los desarrolladores, los operadores y los consumidores de las aplicaciones. Deben tener una identidad para acceder a sus AWS entornos y aplicaciones. Los usuarios humanos que son miembros de su organización (es decir, usted, el desarrollador) se conocen como identidades de personal.

Utilice credenciales temporales al acceder AWS. Puede usar un proveedor de identidad para que sus usuarios humanos proporcionen acceso federado a AWS las cuentas asumiendo funciones, que proporcionan credenciales temporales. Si desea administrar el acceso de manera centralizada, se recomienda utilizar (IAM Identity Center) para administrar el acceso a las cuentas y los permisos de esas cuentas. Para obtener más alternativas, consulte lo siguiente:

- Para obtener más información sobre las prácticas recomendadas, consulte [Prácticas recomendadas de seguridad en IAM](#) en la Guía del usuario de IAM.
- Para crear AWS credenciales de corta duración, consulte [Credenciales de seguridad temporales](#) en la Guía del usuario de IAM.
- Para obtener más información sobre otros proveedores de credenciales de la versión AWS SDK for JavaScript 3, consulte los proveedores de [credenciales estandarizados en la Guía](#) de referencia de herramientas y AWS SDK.

Introducción a Node.js

Esta guía le muestra cómo inicializar un paquete de NPM, agregar un cliente de servicio a su paquete y usar el JavaScript SDK para realizar una acción de servicio.

El escenario

Cree un nuevo paquete de NPM con un archivo principal que haga lo siguiente:

- Crea un bucket de Amazon Simple Storage Service
- Coloca un objeto en el bucket de Amazon S3
- Lee el objeto en el bucket de Amazon S3
- Confirma si el usuario quiere eliminar los recursos

Requisitos previos

Para poder ejecutar el ejemplo, debe hacer lo siguiente:

- Configura la autenticación del SDK. Para obtener más información, consulte [Autenticación de SDK con AWS](#).
- Instale [Node.js](#).

Paso 1: Configurar la estructura de paquetes e instalar los paquetes de cliente

Para configurar la estructura de paquetes e instalar los paquetes de clientes:

1. Cree una nueva carpeta `nodegetstarted` para meter el paquete.
2. Desde la línea de comandos, navegue hasta la nueva carpeta.
3. Ejecute el siguiente comando para crear un archivo `package.json` predeterminado:

```
npm init -y
```

4. Para instalar el paquete de cliente de Amazon S3, ejecute el comando siguiente:

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

5. Añada `"type": "module"` al archivo `package.json`. Esto le indica a Node.js que utilice la sintaxis ESM moderna. El `package.json` final debe ser similar al siguiente:

```
{
  "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"
}
```

Paso 2: Agregar las importaciones y el código de SDK necesarios

Agregue el siguiente código a un archivo llamado `index.js` en la carpeta `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();
}
```

El código de ejemplo se puede encontrar [aquí en GitHub](#)

Paso 3: Ejecutar el ejemplo

Note

¡Recuerde iniciar sesión! Si utiliza el Centro de identidad de IAM para autenticarse, no olvide iniciar sesión con el AWS CLI `aws sso login` comando.

1. Ejecute `node index.js`.
2. Indique si quiere vaciar y eliminar el bucket.
3. Si no elimina el bucket, asegúrese de vaciarlo manualmente y eliminarlo más adelante.

Comenzar en el navegador

En esta sección, se muestra un ejemplo que muestra cómo ejecutar la versión 3 (V3) del SDK para JavaScript en el navegador.

Note

La ejecución de la V3 en el navegador es ligeramente diferente de la versión 2 (V2). Para obtener más información, consulte [Uso de navegadores en la versión 3](#).

Para ver otros ejemplos del uso (V3) del SDK para JavaScript, consulte [Ejemplos de código de SDK para JavaScript \(v3\)](#)

En este ejemplo de aplicación web se muestra lo siguiente:

- Cómo acceder a AWS los servicios mediante Amazon Cognito para la autenticación.
- Cómo leer una lista de objetos en un bucket de Amazon Simple Storage Service (Amazon S3) mediante AWS Identity and Access Management un rol (IAM).

Note

Este ejemplo no se utiliza AWS IAM Identity Center para la autenticación.

El escenario

Amazon S3 es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento líderes del sector. Puede utilizar Amazon S3 para almacenar datos en forma de objetos dentro de contenedores denominados buckets. Para obtener más información sobre Amazon S3, consulte la [Guía del usuario de Amazon S3](#).

En este ejemplo, se muestra cómo configurar y ejecutar una aplicación web que asume un rol de IAM para leer desde un bucket de Amazon S3. El ejemplo utiliza la biblioteca de interfaz de React y las herramientas de interfaz de usuario de Vite para proporcionar un entorno de desarrollo. JavaScript La aplicación web utiliza un grupo de identidades de Amazon Cognito para proporcionar las credenciales necesarias para acceder AWS a los servicios. El ejemplo de código incluido muestra los patrones básicos para cargar y usar el SDK JavaScript en aplicaciones web.

Paso 1: Crear un grupo de identidades de Amazon Cognito y un rol de IAM

En este ejercicio, deberá crear y utilizar un grupo de identidades de Amazon Cognito para proporcionar acceso sin autenticar a su app web para el servicio de Amazon S3. Al crear un grupo de identidades, también se crea un rol AWS Identity and Access Management (IAM) para dar soporte a los usuarios invitados no autenticados. Para este ejemplo, vamos a trabajar con el rol de usuario sin autenticar para concentrarnos en la tarea. Puede integrar la compatibilidad con un proveedor de identidades y los usuarios autenticados más adelante. Para obtener más información sobre como añadir un grupo de identidades de Amazon Cognito, consulte [Tutorial: Crear un grupo de identidades](#) en la Guía para desarrolladores de Amazon Cognito.

Para crear un grupo de identidades de Amazon Cognito y un rol de IAM asociado

1. [Inicie sesión en la consola de Amazon Cognito AWS Management Console y ábrala en https://console.aws.amazon.com/cognito/](https://console.aws.amazon.com/cognito/).
2. En el panel de navegación de la izquierda, elija Grupos de identidades.
3. Elija Crear grupo de identidades.
4. En Configurar la confianza del grupo de identidades, elija Acceso de invitado para la autenticación de usuario.

5. En Configurar permisos, elija Crear un nuevo rol de IAM e introduzca un nombre (por ejemplo, get StartedRole) en el nombre del rol de IAM.
6. En Configurar propiedades, introduzca un nombre (por ejemplo, get StartedPool) en el nombre del grupo de identidades.
7. En Revisar y crear, confirme las selecciones que realizó para el nuevo grupo de identidades. Seleccione Editar para volver al asistente y cambiar cualquier configuración. Cuando haya acabado, seleccione Crear grupo de identidades.
8. Tenga en cuenta el ID del grupo de identidades y la Región del grupo de identidades de Amazon Cognito recién creado. Necesitará estos valores para sustituir *IDENTITY_POOL_ID* y *REGION* en [Paso 4: Configurar el código del navegador](#).

Después de crear el grupo de identidades de Amazon Cognito, ya estará listo para añadir los permisos para Amazon S3 que necesita su app web.

Paso 2: Añadir una política al rol de IAM creado

Para habilitar el acceso a un bucket de Amazon S3 en su aplicación web, utilice el rol de IAM no autenticado (por ejemplo, get StartedRole) creado para su grupo de identidades de Amazon Cognito (por ejemplo, get). StartedPool Para ello, deberá adjuntar una política de IAM al rol. Para obtener más información acerca de cómo modificar roles de IAM, consulte la política [Modificación de los permisos de un rol](#) en la Guía del usuario de IAM.

Añadir una política de Amazon S3 al rol de IAM asociado a usuarios sin autenticar

1. [Inicie sesión en la consola de IAM AWS Management Console y ábrala en https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/).
2. En el panel de navegación izquierdo, seleccione Roles.
3. Elija el nombre del rol que desee modificar (por ejemplo, get StartedRole) y, a continuación, elija la pestaña Permisos.
4. Elija Agregar permisos y luego Adjuntar políticas.
5. En la página Añadir permisos para este rol, busque y, a continuación, active la casilla de verificación de AmazonS3 ReadOnly Access.

Note

Puede utilizar este proceso para habilitar el acceso a cualquier AWS servicio.

6. Elija Añadir permisos.

Tras crear su grupo de identidades de Amazon Cognito y añadir permisos para Amazon S3 a su rol de IAM para usuarios no autenticados, estará listo para añadir y configurar un bucket de Amazon S3.

Paso 3: Agregar un bucket y un objeto de Amazon S3

En este paso, agregará un bucket y un objeto de Amazon S3 para el ejemplo. También habilitará el uso compartido de recursos entre orígenes (CORS) para el bucket. Para obtener más información acerca de la creación de buckets y objetos de Amazon S3, consulte la [Introducción a Amazon S3](#) en la Guía del usuario de Amazon S3.

Añadir un bucket y un objeto de Amazon S3 con CORS

1. Inicie sesión en la consola de Amazon S3 AWS Management Console y ábrala en <https://console.aws.amazon.com/s3/>.
2. En el panel de navegación de la izquierda, elija Buckets y, a continuación, seleccione Crear bucket.
3. Introduzca un nombre de bucket que se ajuste a las [reglas de denominación de buckets](#) (por ejemplo, getstartedbucket) y seleccione Crear bucket.
4. Elija el bucket que ha creado y, a continuación, seleccione la pestaña Objetos. A continuación, elija Cargar.
5. En Archivos y carpetas, elija Añadir archivos.
6. Seleccione un archivo que cargar y luego seleccione Abrir. A continuación, seleccione Cargar para completar la carga del objeto a su bucket.
7. A continuación, seleccione la pestaña Permisos de su bucket y, posteriormente, seleccione Editar en la sección Uso compartido de recursos entre orígenes (CORS). Introduzca el JSON siguiente:

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

```
    "*"
  ],
  "ExposeHeaders": []
}
]
```

8. Elija Guardar cambios.

Tras añadir un bucket de Amazon S3 y un objeto, estará listo para configurar el código del navegador.

Paso 4: Configurar el código del navegador

La aplicación de ejemplo consiste en una aplicación React de una sola página. Los archivos de este ejemplo se encuentran [aquí en GitHub](#).

Configurar la aplicación de ejemplo

1. Instale [Node.js](#).
2. Desde la línea de comandos, clone el [Repositorio de ejemplos de código de AWS](#):

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

3. Desplácese hasta la aplicación de ejemplo:

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

4. Para instalar los paquetes necesarios, ejecute el comando siguiente:

```
npm install
```

5. A continuación, abra `src/App.tsx` en un editor de texto y haga lo siguiente:
 - Sustituya `YOUR_IDENTITY_POOL_ID` por el ID del grupo de identidades de Amazon Cognito que indicó en [Paso 1: Crear un grupo de identidades de Amazon Cognito y un rol de IAM](#).
 - Sustituya el valor de la región por la región asignada a su bucket de Amazon S3 y al grupo de identidades de Amazon Cognito. Tenga en cuenta que las regiones de ambos servicios tienen que ser las mismas (por ejemplo, `us-east-2`).
 - Sustituya `bucket-name` por el nombre del bucket que ha creado en [Paso 3: Agregar un bucket y un objeto de Amazon S3](#).

Una vez que haya reemplazado el texto, guarde el archivo `App.tsx`. Ya tiene todo listo para ejecutar la aplicación web.

Paso 5: Ejecución del ejemplo

Ejecutar la aplicación de ejemplo

1. Desde la línea de comandos, navegue hasta la aplicación de ejemplo:

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

2. Desde la línea de comandos, ejecute el comando siguiente:

```
npm run dev
```

El entorno de desarrollo de Vite se ejecutará con el siguiente mensaje:

```
VITE v4.3.9 ready in 280 ms

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

3. En el navegador web, vaya a la URL que se muestra arriba (por ejemplo, `http://localhost:5173`). La aplicación de ejemplo le mostrará una lista de nombres de archivos de objetos de su bucket de Amazon S3.

Limpieza

Para limpiar los recursos que ha creado en este tutorial, haga lo siguiente:

- En [la consola Amazon S3](#), elimine todos los objetos y buckets creados (por ejemplo, `getstartedbucket`).
- En [la consola de IAM](#), elimine el nombre del rol (por ejemplo, `get StartedRole`).
- En [la consola de Amazon Cognito](#), elimine el nombre del grupo de identidades (por ejemplo, `get StartedPool`).

Configurar el SDK para JavaScript

En los temas de esta sección se explica cómo instalar y cargar el SDK para JavaScript poder acceder a los servicios web compatibles con el SDK.

Note

Los desarrolladores deberían utilizar React Native AWS Amplify para crear nuevos proyectos en AWS. Consulta el [aws-sdk-react-native](#) archivo para obtener más información.

Temas

- [Requisitos previos](#)
- [Instale el SDK para JavaScript](#)
- [Cargue el SDK para JavaScript](#)

Requisitos previos

Instale Node.js en sus servidores, si todavía no lo tiene instalado.

Temas

- [Configure un entorno AWS Node.js](#)
- [Navegadores web compatibles](#)

Configure un entorno AWS Node.js

Para configurar un entorno AWS Node.js en el que pueda ejecutar la aplicación, utilice cualquiera de los métodos siguientes:

- Elija una Imagen de máquina de Amazon (AMI) con Node.js preinstalado. A continuación, cree una instancia Amazon EC2 con esa AMI. Cuando cree la instancia de Amazon EC2, seleccione su AMI en AWS Marketplace. AWS Marketplace Busque Node.js y elija una opción de AMI que incluya una versión preinstalada de Node.js (32 o 64 bits).

- Cree una instancia de Amazon EC2 e instale en ella Node.js. Para obtener más información acerca de cómo instalar Node.js en una instancia de Amazon Linux consulte [Configuración de Node.js en una instancia de Amazon EC2](#).
- Cree un entorno sin servidor mediante la ejecución AWS Lambda de Node.js como una función Lambda. Para obtener más información acerca de cómo utilizar Node.js dentro de una función de Lambda, consulte [Modelos de programación \(Node.js\)](#) en la Guía para desarrolladores de Lambda.
- Implemente su aplicación Node.js en. AWS Elastic Beanstalk Para obtener más información acerca de cómo usar Node.js con Elastic Beanstalk, consulte [Implementación de aplicaciones Node.js en AWS Elastic Beanstalk](#) en la Guía para desarrolladores de AWS Elastic Beanstalk.
- Cree un servidor de aplicaciones Node.js utilizando AWS OpsWorks. Para obtener más información sobre el uso de Node.js con AWS OpsWorks, [consulte Crear la primera pila Node.js](#) en la Guía del AWS OpsWorks usuario.

Navegadores web compatibles

AWS SDK for JavaScript Es compatible con todos los navegadores web modernos.

En la versión 3.183.0 o posterior, el SDK JavaScript utiliza artefactos del ES2020, que son compatibles con las siguientes versiones mínimas.

Navegador	Versión
Google Chrome	80,0 o superior
Mozilla Firefox	80,0 +
Opera	63,0 +
Microsoft Edge	80,0 +
Apple Safari	14.1+
Internet de Samsung	12,0+

En la versión 3.182.0 o anterior, el SDK JavaScript utiliza artefactos de ES5, que admiten las siguientes versiones mínimas.

Navegador	Versión
Google Chrome	49,0 o superior
Mozilla Firefox	45,0 +
Opera	36,0 +
Microsoft Edge	12,0+
Windows Internet Explorer	N/A
Apple Safari	9.0+
Navegador Android	76,0 +
Browser UC	12,12 +
Internet de Samsung	5,0+

Note

AWS Amplify Es posible que marcos como este no ofrezcan la misma compatibilidad con el navegador que el SDK. JavaScript Consulte la [Documentación de AWS Amplify](#) para obtener más detalles.

Instale el SDK para JavaScript

No todos los servicios están disponibles de forma inmediata en el SDK ni en todas AWS las regiones.

Para instalar un servicio desde [npm](#), [el AWS SDK for JavaScript administrador de paquetes de Node.js](#), introduzca el siguiente comando en la línea de comandos, donde **SERVICE** es el nombre de un servicio, por ejemplo. s3

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

Para obtener una lista completa de los paquetes de clientes de AWS SDK for JavaScript servicios, consulta la [guía de referencia de la AWS SDK for JavaScript API](#).

Cargue el SDK para JavaScript

Después de instalar el SDK, puede cargar un paquete de clientes en su aplicación de nodo usando `import`. Por ejemplo, para cargar el cliente Amazon S3 y el [ListBuckets](#) comando Amazon S3, utilice lo siguiente.

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

Configurar el SDK para JavaScript

Antes de usar el SDK para JavaScript invocar servicios web mediante la API, debe configurar el SDK. Como mínimo, tiene que configurar:

- La AWS región en la que solicitará los servicios
- Cómo se autentica su código con AWS

Además de configurar estos valores, es posible que también tenga que configurar permisos para sus recursos de AWS. Por ejemplo, puede limitar el acceso a un bucket de Amazon S3 o restringir una tabla de Amazon DynamoDB para acceso de solo lectura.

La [guía de referencia de AWS los SDK y las herramientas](#) también contiene configuraciones, funciones y otros conceptos fundamentales comunes a muchos de los SDK. AWS

En los temas de esta sección se describen las formas de configurar el SDK JavaScript para Node.js y su JavaScript ejecución en un navegador web.

Temas

- [Configuración según el servicio](#)
- [Establezca la región AWS](#)
- [Establece las credenciales](#)
- [Consideraciones de Node.js](#)
- [Consideraciones sobre los scripts de navegador](#)

Configuración según el servicio

Puede configurar el SDK pasando la información de configuración a un objeto de servicio.

La configuración a nivel de servicio proporciona un control significativo sobre los servicios individuales, lo que le permite actualizar la configuración de los objetos de servicio individuales cuando sus necesidades difieran de la configuración predeterminada.

Note

En la versión 2.x, la configuración del AWS SDK for JavaScript servicio se podía pasar a los constructores de clientes individuales. Sin embargo, estas configuraciones primero se fusionarían automáticamente en una copia de la configuración global del SDK `AWS.config`. Además, llamar a `AWS.config.update({/* params */})` solo actualizaba la configuración de los clientes de servicio instanciados después de realizar la llamada de actualización, no a los clientes existentes.

Este comportamiento era una fuente frecuente de confusión y dificultaba añadir una configuración al objeto global que solo afectara a un subconjunto de clientes de servicio de forma compatible con versiones posteriores. En la versión 3, ya no existe una configuración global gestionada por el SDK. La configuración tiene que pasarse a cada cliente de servicio del que se cree una instancia. Sigue siendo posible compartir la misma configuración entre varios clientes, pero esa configuración no se fusionará automáticamente con un estado global.

Establezca la configuración por servicio

Se accede a cada servicio que utilices en el SDK a través de un objeto de servicio que forma parte de la API de ese servicio. JavaScript Por ejemplo, para acceder al servicio Amazon S3, debe crear el objeto de servicio Amazon S3. Puede especificar los valores de configuración que son específicos de un servicio como parte del constructor para dicho objeto de servicio.

Por ejemplo, si necesita acceder a los objetos de Amazon EC2 en varias AWS regiones, cree un objeto de servicio de Amazon EC2 para cada región y, a continuación, establezca la configuración regional de cada objeto de servicio en consecuencia.

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

Establezca la región AWS

Una AWS región es un conjunto de AWS recursos con nombre asignado en la misma área geográfica. Un ejemplo de región es `us-east-1`, que es la región Este de EE. UU. (Norte de Virginia). Al crear un cliente de servicio en el SDK, se especifica una región JavaScript para que

el SDK acceda al servicio de esa región. Algunos servicios de solo están disponibles en regiones específicas.

El SDK para JavaScript no selecciona una región de forma predeterminada. Sin embargo, puede configurar la AWS región mediante una variable de entorno o un `config` archivo de configuración compartido.

En un constructor de clase de cliente

Al crear una instancia de un objeto de servicio, puede especificar la AWS región de ese recurso como parte del constructor de la clase de cliente, como se muestra aquí.

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

Usa una variable de entorno

Puede establecer la región mediante la variable de entorno `AWS_REGION`. Si defines esta variable, el SDK la JavaScript lee y la usa.

Usa un archivo de configuración compartido

Del mismo modo que el archivo de credenciales compartidas te permite almacenar las credenciales para que las utilice el SDK, puedes guardar tu AWS región y otros ajustes de configuración en un archivo compartido con el nombre `config` del SDK que vas a usar. Si la variable de `AWS_SDK_LOAD_CONFIG` entorno se establece en un valor verdadero, el SDK busca JavaScript automáticamente un `config` archivo cuando se carga. La ubicación donde guarde el archivo `config` depende de su sistema operativo:

- Usuarios de Linux, macOS o Unix: `~/.aws/config`
- Usuarios de Windows: `C:\Users\USER_NAME\.aws\config`

Si todavía no tiene un archivo `config` compartido, puede crear uno en el directorio designado. En el siguiente ejemplo, el archivo `config` establece la región y el formato de salida.

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

Para obtener más información sobre el uso de archivos de `config` y `credentials` compartidos, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

Orden de prioridad para establecer la región

El orden de prioridad para la configuración de una región es el siguiente:

1. Si se transfiere una región a un constructor de clase de cliente, se usa dicha región.
2. Si se establece una región en la variable de entorno, se utiliza esa región.
3. De lo contrario, se utiliza la región definida en el archivo de configuración compartido.

Establece las credenciales

AWS usa las credenciales para identificar quién llama a los servicios y si se permite el acceso a los recursos solicitados.

Ya sea que se ejecute en un navegador web o en un servidor Node.js, el JavaScript código debe obtener credenciales válidas antes de poder acceder a los servicios a través de la API. Las credenciales se pueden configurar por servicio, por medio de la transferencia directa de las credenciales a un objeto de servicio.

Hay varias formas de configurar las credenciales que difieren entre Node.js y JavaScript en los navegadores web. En los temas de esta sección se describe cómo configurar credenciales en Node.js o en navegadores web. En cada caso, las opciones se presentan en el orden recomendado.

Prácticas recomendadas para las credenciales

Configurar correctamente las credenciales garantiza que su aplicación o script de navegador pueda obtener acceso a los servicios y los recursos necesarios y, al mismo tiempo, minimiza su exposición a problemas de seguridad que puedan repercutir en aplicaciones críticas o pongan en peligro información confidencial.

Cuando se configuran credenciales es importante aplicar siempre el principio de conceder el privilegio mínimo necesario para llevar a cabo la tarea. Es más seguro proporcionar permisos mínimos sobre sus recursos y añadir más permisos según sea necesario, en lugar de proporcionar permisos que superan el privilegio mínimo y luego tener que solucionar problemas de seguridad que puedan aparecer más tarde. Por ejemplo, a menos que necesite leer y escribir recursos individuales,

como objetos en un bucket de Amazon S3 o una tabla de DynamoDB, establezca dichos permisos en solo lectura.

Para obtener más información sobre la concesión de privilegios mínimos, consulte la sección [Conceder privilegios mínimos](#) del tema de prácticas recomendadas de la Guía del usuario de IAM.

Temas

- [Establecer credenciales en Node.js](#)
- [Establecer credenciales en un navegador web](#)

Establecer credenciales en Node.js

Recomendamos a los nuevos usuarios que desarrollen sus proyectos de forma local y que no cuenten con un método de autenticación de su empresa para realizar la configuración AWS IAM Identity Center. Para obtener más información, consulte [Autenticación de SDK con AWS](#).

En Node.js hay varias formas de proporcionar sus credenciales al SDK. Algunas de ellas son más seguras y otras ofrecen más comodidad mientras se desarrolla una aplicación. Al obtener las credenciales en Node.js, sea especialmente cuidadoso si se basa en más de un origen como, por ejemplo, una variable de entorno y un archivo JSON que carga. Puede cambiar los permisos con los que se ejecuta su código sin darse cuenta de que se ha producido el cambio.

AWS SDK for JavaScript La versión 3 proporciona una cadena de proveedores de credenciales predeterminada en Node.js, por lo que no es necesario que proporciones un proveedor de credenciales de forma explícita. La [cadena de proveedores de credenciales](#) predeterminada intenta resolver las credenciales de una variedad de fuentes diferentes con una prioridad determinada, hasta que una de las fuentes devuelva una credencial. [Puede encontrar la cadena de proveedores de credenciales del SDK para la versión 3 aquí. JavaScript](#)

Cadena de proveedores de credenciales

Todos los SDK tienen una serie de lugares (o fuentes) que consultan para obtener credenciales válidas y utilizarlas para realizar una solicitud a un Servicio de AWS. Una vez que se encuentran las credenciales válidas, se detiene la búsqueda. Esta búsqueda sistemática se denomina cadena predeterminada de proveedores de credenciales.

Para cada paso de la cadena, hay diferentes maneras de establecer los valores. La configuración de valores directamente en el código siempre tiene prioridad, seguida de la configuración como

variables de entorno y, por último, en el archivo compartido. AWS config Para obtener más información, consulte [Prioridad de configuración](#) en la Guía de referencia de las herramientas y los SDK de AWS .

La guía de referencia de AWS los SDK y las herramientas contiene información sobre los ajustes de configuración del SDK que utilizan todos los AWS SDK y los. AWS CLI Para obtener más información sobre cómo configurar el SDK a través del AWS config archivo compartido, consulte Archivos de [credenciales y configuración compartidos](#). Para obtener más información sobre cómo configurar el SDK mediante la configuración de variables de entorno, consulte [Compatibilidad con variables de entorno](#).

Para autenticarse AWS, AWS SDK for JavaScript comprueba los proveedores de credenciales en el orden que se indica en la siguiente tabla.

AWS SDK for JavaScript La API hace referencia al método del proveedor de credenciales por prioridad	Proveedores de credenciales disponibles	AWS Guía de referencia de SDK y herramientas
fromEnv()	AWS claves de acceso desde variables de entorno	AWS claves de acceso
fromSSO()	AWS IAM Identity Center. En esta guía, consulte Autenticación de SDK con AWS .	Proveedor de credenciales del IAM Identity Center
fromIni()	AWS claves de acceso desde <code>credentials</code> archivos <code>config</code> y archivos compartidos	AWS claves de acceso
	Proveedor de entidades de confianza (como <code>AWS_ROLE_ARN</code>)	Asumir un rol de IAM
	token de identidad web de AWS Security Token Service (AWS STS)	Federe con identidad web u OpenID Connect

AWS SDK for JavaScript La API hace referencia al método del proveedor de credenciales por prioridad	Proveedores de credenciales disponibles	AWS Guía de referencia de SDK y herramientas
	Credenciales de Amazon Elastic Container Service (Amazon ECS)	Proveedor de credenciales de contenedor
	Credenciales de perfil de instancia de Amazon Elastic Compute Cloud (Amazon EC2) (proveedor de credenciales IMDS)	Proveedor de credenciales IMDS
	Proveedor de credenciales de proceso	Proveedor de credenciales de proceso
	AWS IAM Identity Center credenciales	Proveedor de credenciales del IAM Identity Center
fromProcess()	Proveedor de credenciales de proceso	Proveedor de credenciales de proceso
fromTokenFile()	token de identidad web de AWS Security Token Service (AWS STS)	Federe con identidad web u OpenID Connect
fromContainerMetadata()	Credenciales de Amazon Elastic Container Service (Amazon ECS)	Proveedor de credenciales de contenedor
fromInstanceMetadata()	Credenciales de perfil de instancia de Amazon Elastic Compute Cloud (Amazon EC2) (proveedor de credenciales IMDS)	Proveedor de credenciales IMDS

Si siguió el enfoque recomendado para los nuevos usuarios para empezar, configuró la autenticación AWS Single Sign-On en [Autenticación de SDK con AWS](#) del tema Introducción. Otros métodos de autenticación son útiles en diferentes situaciones. Para evitar riesgos de seguridad, recomendamos utilizar siempre credenciales a corto plazo. Para conocer otros procedimientos de métodos de autenticación, consulte [Autenticación y acceso](#) en la Guía de referencia de las herramientas y los SDK de AWS.

En los temas de esta sección se describe cómo cargar credenciales en Node.js.

Temas

- [Cargue las credenciales en Node.js desde las funciones de IAM para Amazon EC2](#)
- [Cargar credenciales para una función Lambda de Node.js](#)

Cargue las credenciales en Node.js desde las funciones de IAM para Amazon EC2

Si ejecuta su aplicación de Node.js en una instancia de Amazon EC2, puede utilizar roles de IAM para Amazon EC2 para suministrar automáticamente credenciales a la instancia. Si configura la instancia para que utilice roles de IAM, el SDK selecciona automáticamente las credenciales de IAM para su aplicación, lo que elimina la necesidad de proporcionar las credenciales manualmente.

Para obtener más información sobre cómo añadir roles de IAM a una instancia de Amazon EC2, consulte [Roles de IAM para Amazon EC2](#).

Cargar credenciales para una función Lambda de Node.js

Al crear una AWS Lambda función, debe crear una función de IAM especial que tenga permiso para ejecutar la función. Este rol se denomina el rol de ejecución. Cuando configura una función de Lambda, tiene que especificar el rol de IAM que ha creado como el rol de ejecución correspondiente.

El rol de ejecución proporciona a la función de Lambda las credenciales que necesita para ejecutarse y para invocar otros servicios web. Como resultado, no es necesario proporcionar credenciales al código Node.js que escribe dentro de una función de Lambda.

Para obtener más información sobre cómo crear un rol de ejecución de Lambda, consulte [Administración de permisos: usar un rol de IAM \(rol de ejecución\)](#) en la Guía para desarrolladores de Lambda.

Establecer credenciales en un navegador web

Hay varias formas de proporcionar credenciales al SDK desde scripts de navegador. Algunas de ellas son más seguras y otras ofrecen más comodidad mientras se desarrolla un script.

A continuación se muestran las formas de suministro de credenciales según su orden de recomendación:

1. Uso de Amazon Cognito para autenticar usuarios y suministrar credenciales
2. Uso de identidades federadas web

Warning

No recomendamos codificar sus AWS credenciales de forma rígida en sus scripts. Si las codifica de forma rígida, corre el riesgo de que su ID de clave de acceso y la clave de acceso secreta queden expuestos.

Temas

- [Utilice Amazon Cognito Identity para autenticar a los usuarios](#)

Utilice Amazon Cognito Identity para autenticar a los usuarios

La forma recomendada de obtener AWS credenciales para los scripts del navegador es utilizar el cliente de credenciales de Amazon Cognito Identity. `CognitoIdentityClient` Amazon Cognito permite la autenticación de los usuarios a través de proveedores de identidad de terceros.

Para usar Amazon Cognito Identity, primero debe crear un grupo de identidades en la consola Amazon Cognito. Un grupo de identidades representa el grupo de identidades que su aplicación proporciona a los usuarios. Las identidades que se dan a los usuarios identifican de forma inequívoca cada cuenta de usuario. Las identidades de Amazon Cognito no son credenciales. Se intercambian por credenciales mediante el soporte de federación de identidades web en AWS Security Token Service (AWS STS).

Amazon Cognito es útil para administrar la abstracción de identidades a través de varios proveedores de identidades. La identidad que se carga se intercambia por credenciales en AWS STS.

Configurar el objeto de credenciales de Amazon Cognito Identity

Si todavía no ha creado un grupo de identidades, cree uno para usarlo con los scripts de navegador en la [consola Amazon Cognito](#) antes de configurar su cliente de Amazon Cognito. Cree y asocie roles de IAM autenticados y sin autenticar para su grupo de identidades. Para obtener más información, consulte [Tutorial: Crear un grupo de identidades](#) en la Guía para desarrolladores de Amazon Cognito.

La identidad de los usuarios sin autenticar no se verifica, lo que hace que este rol sea adecuado para los usuarios invitados de la aplicación o para cuando no importa si se ha verificado la identidad de los usuarios. Los usuarios autenticados inician sesión en la aplicación a través de un proveedor de identidades externo que verifica sus identidades. Asegúrese de asignar los permisos de los recursos de forma adecuada, para no conceder acceso a ellos a los usuarios no autenticados.

Tras configurar un grupo de identidades, utilice el método `fromCognitoIdentityPool` de `@aws-sdk/credential-providers` para recuperar las credenciales del grupo de identidades. En el siguiente ejemplo de creación de un cliente de Amazon S3, sustituya `AWS_REGION` por la región y `IDENTITY_POOL_ID` por el ID del grupo 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.
    },
  })
});
```

La propiedad opcional `logins` es un mapeo entre los nombres de los proveedores de identidad y los tokens de identidad de los proveedores. La forma de obtener el token del proveedor de identidad depende del proveedor que se utilice. Por ejemplo, si utiliza un grupo de usuarios de Amazon

Cognito como proveedor de autenticación, puede utilizar un método similar al que se muestra a continuación.

```
// 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;
};
```

Cambie los usuarios no autenticados a usuarios autenticados

Amazon Cognito es compatible con los usuarios autenticados y no autenticados. Los usuarios sin autenticar reciben acceso a sus recursos incluso si no han iniciado sesión con alguno de sus proveedores de identidades. Este grado de acceso es útil para mostrar contenido a usuarios antes de que inicien sesión. Cada usuario sin autenticar tiene una identidad única en Amazon Cognito, aunque no haya iniciado sesión ni se haya autenticado individualmente.

Usuario sin autenticar inicialmente

Los usuarios suelen comenzar con el rol sin autenticar, para el que se establece la propiedad de credenciales de su objeto de configuración sin una propiedad `logins`. En este caso, las credenciales predeterminadas podrían tener el siguiente aspecto:

```
// 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.
});
```

Cambie a usuario autenticado

Cuando un usuario sin autenticar inicia sesión en un proveedor de identidades y tiene un token, puede cambiar el usuario de no estar autenticado a estar autenticado llamando a una función personalizada que actualiza el objeto de credenciales y añade el 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;
}
```

Consideraciones de Node.js

Aunque el código Node.js sí JavaScript lo es, su uso AWS SDK for JavaScript en Node.js puede diferir del uso del SDK en los scripts del navegador. Algunos métodos de API funcionan en Node.js pero no en scripts de navegador y viceversa. Y el uso correcto de algunas API depende de su conocimiento de patrones de codificación de Node.js habituales como la importación y el uso de otros módulos de Node.js como el módulo File System (fs).

Utilice los módulos Node.js integrados

Node.js ofrece un conjunto de módulos integrados que puede utilizar sin necesidad de instalarlos. Para utilizar estos módulos, cree un objeto con el método `require` para especificar el nombre del módulo. Por ejemplo, para incluir el módulo HTTP integrado, utilice el código siguiente.

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

Invoque métodos del módulo como si fueran métodos de dicho objeto. Por ejemplo, a continuación le mostramos código que lee un archivo 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 obtener una lista completa de todos los módulos integrados que Node.js proporciona, consulte la [documentación de Node.js](#) sobre el sitio web de Node.js.

Utilice paquetes npm

Además de los módulos integrados, también puede incluir e incorporar código de terceros desde npm, el administrador de paquetes de Node.js. Se trata de un repositorio de paquetes Node.js de código abierto y una interfaz de línea de comandos para instalar dichos paquetes. Para obtener más información acerca de npm y una lista de los paquetes disponibles actualmente, consulte <https://www.npmjs.com>. También puede obtener información sobre los paquetes adicionales de Node.js que puede usar [aquí](#). GitHub

Configurar MaxSockets en Node.js

En Node.js, puede definir el número máximo de conexiones por origen. Si `maxSockets` está establecido, el cliente HTTP de bajo nivel pone en cola las solicitudes y las asigna a conectores a medida que estos están disponibles.

Esto le permite configurar un límite máximo del número de solicitudes simultáneas a un determinado origen a la vez. Si disminuye este valor, podrá reducir el número de errores de tiempo de espera o de limitación controlada. Sin embargo, también puede aumentar el uso de la memoria, ya que las solicitudes se ponen en cola hasta que un conector esté disponible.

En el siguiente ejemplo, se muestra cómo se establece `maxSockets` para un cliente de 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
  });
});
```

El SDK JavaScript utiliza un `maxSockets` valor de 50 si no se proporciona un valor o un `Agent` objeto. Si proporciona un `Agent` objeto, se utilizará su `maxSockets` valor. Para obtener más información sobre la configuración `maxSockets` de Node.js, consulte la [documentación de Node.js](#).

A partir de la versión 3.521.0 del AWS SDK for JavaScript, puede utilizar la siguiente sintaxis [abreviada](#) para configurar `requestHandler`

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

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

Reutilice las conexiones con keep-alive en Node.js

El agente HTTP o HTTPS predeterminado de Node.js crea una nueva conexión TCP para cada nueva solicitud. Para evitar el coste de establecer una nueva conexión, AWS SDK for JavaScript reutiliza las conexiones TCP de forma predeterminada.

En el caso de las operaciones de corta duración, como las consultas de Amazon DynamoDB, la sobrecarga en latencia de la configuración de una conexión TCP puede ser mayor que la propia operación. Además, dado que el [cifrado en reposo de DynamoDB](#) está integrado, es posible que se [AWS KMS](#) produzcan latencias en la base de datos al tener que restablecer AWS KMS nuevas entradas de caché para cada operación.

Si no desea reutilizar las conexiones TCP, puede deshabilitar la reutilización activa de estas conexiones para cada cliente de servicio, como se muestra en el siguiente ejemplo para un cliente de DynamoDB. `keepAlive`

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

Si `keepAlive` está habilitado, también puede establecer el retraso inicial de los paquetes TCP Keep-Alive con `keepAliveMsecs`, que de forma predeterminada es 1000 ms. Consulte la [documentación de Node.js](#) para obtener más detalles.

Configure los proxies para Node.js

Si no puedes conectarte directamente a Internet, el SDK JavaScript admite el uso de proxies HTTP o HTTPS a través de un agente HTTP de terceros.

Para encontrar un agente HTTP de terceros, busque “proxy HTTP” en [npm](#).

Para instalar un proxy de agente HTTP de terceros, introduzca lo siguiente en la línea de comandos, donde **PROXY** es el nombre del paquete de npm.

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

Para usar un proxy en su aplicación, utilice la propiedad `httpAgent` y `httpsAgent`, como se muestra en el siguiente ejemplo para un cliente de 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` no es lo mismo que `httpsAgent`, y dado que la mayoría de las llamadas del cliente serán a `https`, ambas deberían estar configuradas.

Registre paquetes de certificados en Node.js

Los almacenes de confianza de Node.js incluyen los certificados necesarios para tener acceso a los servicios de AWS. En algunos casos, puede ser preferible incluir únicamente un conjunto específico de certificados.

En este ejemplo, se usa un certificado específico en el disco para crear un `https.Agent` que rechace las conexiones a menos que se proporcione el certificado designado. A continuación, el cliente de DynamoDB utiliza el `https.Agent` que se acaba de crear.

```
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  
  })  
});
```

Consideraciones sobre los scripts de navegador

En los temas siguientes se describen las consideraciones especiales que se deben tener en cuenta a la hora de utilizarlos AWS SDK for JavaScript en los scripts del navegador.

Temas

- [Cree el SDK para navegadores](#)
- [Uso compartido de recursos entre orígenes \(CORS\)](#)
- [Combine aplicaciones con webpack](#)

Cree el SDK para navegadores

A diferencia del SDK de la JavaScript versión 2 (V2), la versión 3 no se proporciona como un JavaScript archivo con soporte incluido para un conjunto de servicios predeterminado. En cambio, la versión 3 le permite agrupar e incluir en el navegador solo el SDK para los JavaScript archivos que necesite, lo que reduce la sobrecarga. Recomendamos usar Webpack para agrupar el SDK necesario para JavaScript los archivos y cualquier paquete adicional de terceros que necesite en un solo Javascript archivo y cargarlo en los scripts del navegador mediante una `<script>` etiqueta. Para obtener más información sobre Webpack, consulte [Combine aplicaciones con webpack](#). Para ver un ejemplo en el que se usa Webpack para cargar el SDK de la versión 3 JavaScript en un navegador, consulte. [Crear una aplicación para enviar datos a DynamoDB](#)

Si trabaja con el SDK fuera de un entorno que aplique el CORS en su navegador y si desea acceder a todos los servicios que proporciona el SDK JavaScript, puede crear una copia personalizada del SDK de forma local clonando el repositorio y ejecutando las mismas herramientas de compilación que crean la versión alojada predeterminada del SDK. En las siguientes secciones se describen los pasos necesarios para compilar el SDK con servicios adicionales y versiones de API.

Usa el SDK Builder para crear el SDK para JavaScript

Note

Amazon Web Services versión 3 (V3) ya no es compatible con Browser Builder. Para minimizar el uso del ancho de banda de las aplicaciones del navegador, le recomendamos que importe los módulos con nombre asignado y los agrupe para reducir su tamaño. Para obtener más información sobre la agrupación, consulte [Combine aplicaciones con webpack](#).

Uso compartido de recursos entre orígenes (CORS)

El uso compartido de recursos entre orígenes o CORS es una característica de seguridad de los navegadores web modernos. Habilita a los navegadores web para que puedan negociar qué dominios pueden realizar solicitudes de sitios web o servicios externos.

CORS es un factor importante que hay que tener en cuenta cuando se desarrollan aplicaciones de navegador con AWS SDK for JavaScript, ya que la mayoría de las solicitudes a recursos se envían a un dominio externo como, por ejemplo, el punto de enlace de un servicio web. Si su JavaScript entorno aplica la seguridad de CORS, debe configurar CORS con el servicio.

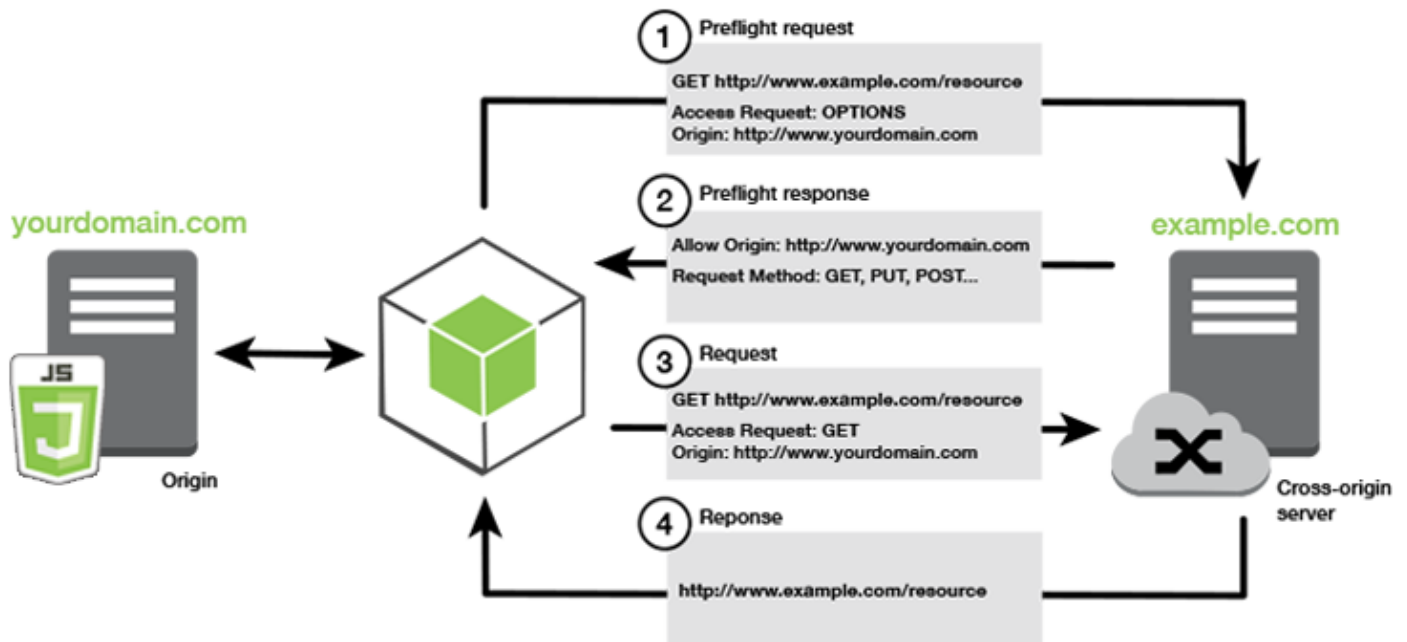
CORS determina si se permitirá el uso compartido de recursos en una solicitud entre orígenes basándose en lo siguiente:

- El dominio específico que efectúa la solicitud.
- El tipo de solicitud HTTP que se realiza (GET, PUT, POST, DELETE, etc.).

Funcionamiento de CORS

En el caso más sencillo, el navegador script realiza una solicitud GET para obtener un recurso de un servidor que se encuentra en otro dominio. En función de cómo sea la configuración de CORS de dicho servidor, si la solicitud proviene de un dominio con permiso para enviar solicitudes GET, el servidor de orígenes cruzados responderá devolviendo el recurso que se ha solicitado.

Si el dominio de solicitud o el tipo de solicitud HTTP no está autorizado, se denegará la solicitud. Sin embargo, con CORS es posible realizar una solicitud preliminar antes de enviarla realmente. En dicho caso, se realiza una solicitud preliminar en la que se envía la operación de solicitud de acceso OPTIONS. Si la configuración CORS del servidor de origen otorga acceso al dominio que realiza la solicitud, el servidor enviará una respuesta preliminar que contenga una lista de todos los tipos de solicitud HTTP que el dominio que realiza la solicitud puede hacer al recurso solicitado.



¿Se requiere la configuración de CORS?

Es preciso configurar CORS en los buckets de Amazon S3 para poder realizar operaciones en ellos. En algunos JavaScript entornos, es posible que el CORS no se aplique y, por lo tanto, no sea necesario configurarlo. Por ejemplo, si aloja su aplicación de un bucket de Amazon S3 y obtiene acceso a recursos de `*.s3.amazonaws.com` o algún otro punto de conexión específico, sus solicitudes no tendrán acceso a un dominio externo. Por lo tanto, esta configuración no necesita CORS. En este caso, se seguirá utilizando CORS para servicios ajenos a Amazon S3.

Configurar CORS para un bucket de Amazon S3

Puede configurar un un bucket de Amazon S3 para utilizar CORS en la consola de Amazon S3.

Si está configurando CORS en la consola de administración de servicios AWS web, debe usar JSON para crear una configuración de CORS. La nueva consola de administración de servicios AWS web solo admite configuraciones CORS de JSON.

⚠ Important

En la nueva consola de administración de servicios AWS web, la configuración CORS debe ser JSON.

1. En la consola de administración de servicios AWS web, abra la consola de Amazon S3, busque el bucket que desee configurar y active su casilla de verificación.
2. En el panel que se abre, seleccione Permisos.
3. En la pestaña Permisos, elija Configuración de CORS.
4. Escriba su configuración de CORS en el Editor de configuración de CORS y, a continuación, seleccione Guardar.

Una configuración de CORS es un archivo XML que contiene una serie de reglas dentro de una `<CORSRule>`. Una configuración puede tener hasta 100 reglas. Una regla se define con una de las siguientes etiquetas:

- `<AllowedOrigin>`: especifica orígenes de dominios a los que permite realizar solicitudes de dominio cruzado.
- `<AllowedMethod>`: especifica un tipo de solicitud que permite (GET, PUT, POST, DELETE, HEAD) en solicitudes de dominio cruzado.
- `<AllowedHeader>`: especifica los encabezados que están permitidos en una solicitud preliminar.

Para ver las configuraciones de ejemplo, consulte [Cómo configurar CORS en mi bucket](#) en la Guía del usuario de Amazon Simple Storage Service.

Ejemplo de configuración de CORS

El siguiente ejemplo de configuración de CORS permite a un usuario ver, añadir, eliminar o actualizar los objetos contenidos en un bucket del dominio `example.org`. Sin embargo, le recomendamos que mantenga `<AllowedOrigin>` en el dominio de su sitio web. Puede especificar "*" para permitir cualquier origen.

Important

En la nueva consola de S3, la configuración de CORS debe ser JSON.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSSConfiguration 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"
    ]
  }
]
```

Esta configuración no autoriza al usuario a realizar acciones en el bucket. Habilita al modelo de seguridad del navegador para que permita una solicitud a Amazon S3. Los permisos tienen que configurarse a través de permisos de bucket o permisos de rol de IAM.

Puede utilizar `ExposeHeader` para permitir que el SDK lea los encabezados de respuestas que devuelve Amazon S3. Por ejemplo, para leer el encabezado `ETag` de una operación `PUT` o una carga

multiparte, debe incluir la etiqueta `ExposeHeader` en su configuración, tal y como se muestra en el ejemplo anterior. El SDK solo puede obtener acceso a los encabezados que se exponen a través de la configuración de CORS. Si establece metadatos en el objeto, los valores se devuelven como encabezados con el prefijo `x-amz-meta-`, como `x-amz-meta-my-custom-header`, y también deben exponerse de la misma manera.

Combine aplicaciones con webpack

El uso de módulos de código por parte de aplicaciones web en scripts de navegador o Node.js crea dependencias. Estos módulos de código pueden tener dependencias propias, lo que se traduce en una colección de módulos interconectados que su aplicación necesita para funcionar. Para administrar dependencias, puede utilizar un instalador de módulos como Webpack.

El instalador de módulos Webpack analiza el código de la aplicación para buscar instrucciones `import` o `require` y crear agrupaciones que contengan todos los recursos que la aplicación necesita. Esto es para que los activos se puedan servir fácilmente a través de una página web. El SDK para JavaScript puede incluir webpack como una de las dependencias para incluirlo en el paquete de salida.

Para obtener más información al respecto de webpack, consulte el paquete de [módulos webpack](#) en GitHub.

Instale webpack

Para instalar el instalador de módulos Webpack, en primer lugar debe tener npm, el administrador de paquetes de Node.js, instalado. Escriba el siguiente comando para instalar la webpack CLI y el JavaScript módulo.

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

Para usar el módulo `path` para trabajar con rutas de archivos y directorios, que se instala automáticamente con Webpack, es posible que necesite instalar el paquete `path-browserify` de Node.js.

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

Configure el paquete web

De forma predeterminada, Webpack busca un JavaScript archivo con un nombre `webpack.config.js` en el directorio raíz de tu proyecto. Este archivo especifica sus opciones de

configuración. El siguiente es un ejemplo de un archivo de `webpack.config.js` configuración para la WebPack versión 5.0.0 y versiones posteriores.

Note

Los requisitos de configuración de Webpack varían según la versión de Webpack que instale. Para obtener más información, consulte la [documentación de 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"}]
 * }
 */
};
```

En este ejemplo, `browser.js` se especifica como el punto de entrada. El punto de entrada es el archivo que Webpack utiliza para comenzar a buscar módulos importados. El nombre de archivo de la salida se especifica como `bundle.js`. Este archivo de salida contendrá todo lo que JavaScript la aplicación necesita para ejecutarse. Si el código especificado en el punto de entrada importa o requiere otros módulos, como el SDK JavaScript, ese código se incluye sin necesidad de especificarlo en la configuración.

Ejecute webpack

Para desarrollar una aplicación para utilizar Webpack, añada lo siguiente al objeto `scripts` de su archivo `package.json`.

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

El siguiente es un archivo `package.json` de ejemplo que muestra cómo agregar 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 compilar su aplicación, introduzca el comando siguiente.

```
npm run build
```

Luego, el paquete de webpack módulos genera el JavaScript archivo que especificó en el directorio raíz de su proyecto.

Usa el paquete webpack

Para utilizar la agrupación en un script de navegador, puede incorporar la agrupación con una etiqueta `<script>`, tal y como se muestra en el siguiente ejemplo.

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

Paquete para Node.js

Puede utilizar Webpack para generar agrupaciones que se ejecutan en Node.js especificando node como destino en la configuración.

```
target: "node"
```

Esto resulta útil al ejecutar una aplicación de Node.js en un entorno donde el espacio en el disco es limitado. A continuación se muestra un ejemplo de configuración de `webpack.config.js` con Node.js especificado como destino de salida.

```
// 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"}]
}
**/
};
```

Trabaje con AWS los servicios del SDK para JavaScript

La AWS SDK for JavaScript versión 3 proporciona acceso a los servicios que admite a través de una colección de clases de clientes. A partir de estas clases del cliente, se crean objetos de interfaz de servicios, normalmente denominados objetos de servicio. Cada AWS servicio compatible tiene una o más clases de clientes que ofrecen API de bajo nivel para utilizar las funciones y los recursos del servicio. Por ejemplo, las API de Amazon DynamoDB están disponibles a través de la clase DynamoDB.

Los servicios incluidos en el SDK JavaScript siguen el patrón de solicitud-respuesta para intercambiar mensajes con las aplicaciones que realizan llamadas. En este patrón, el código que invoca un servicio envía una solicitud HTTP/HTTPS a un punto de enlace para el servicio. La solicitud contiene parámetros necesarios para invocar correctamente la característica específica a la que se llama. El servicio que se invoca genera una respuesta que se devuelve al solicitante. La respuesta contiene datos si la operación se ha realizado correctamente o bien contiene información de error si dicha operación ha generado errores.

La invocación de un AWS servicio incluye todo el ciclo de vida de solicitud y respuesta de una operación en un objeto de servicio, incluidos los intentos de reintento. Una solicitud contiene cero o más propiedades como parámetros de JSON. La respuesta se encapsula en un objeto relacionado con la operación y se devuelve al solicitante mediante una de varias técnicas, como una función de devolución de llamada o una promesa. JavaScript

Temas

- [Crea y llama a objetos de servicio](#)
- [Llame a los servicios de forma asíncrona](#)
- [Cree solicitudes de clientes de servicio](#)
- [Gestione las respuestas de los clientes del servicio](#)
- [Trabaja con JSON](#)
- [SDK para ejemplos JavaScript de código](#)

Crea y llama a objetos de servicio

La JavaScript API es compatible con la mayoría de AWS los servicios disponibles. Cada servicio de la JavaScript API proporciona una clase de cliente con un send método que se utiliza para invocar

todas las API compatibles con el servicio. Para obtener más información sobre las clases de servicio, las operaciones y los parámetros de la JavaScript API, consulta la [referencia de la API](#).

Cuando utiliza el SDK en Node.js, añada el paquete del SDK para cada servicio que necesita a su aplicación mediante `import`, que ofrece soporte para todos los servicios actuales. En el siguiente ejemplo se crea un objeto de servicio de Amazon S3 en la región de `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 los parámetros del objeto de servicio

Al llamar a un método de un objeto de servicio, transfiera los parámetros en JSON según los requiera la API. Por ejemplo, en Amazon S3, para obtener un objeto para un bucket y una clave específicos, pase los siguientes parámetros al `GetObjectCommand` método desde `S3Client`. Para obtener más información acerca de cómo transferir parámetros JSON, consulte [Trabaja con JSON](#).

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

Para obtener más información sobre los parámetros de Amazon S3, consulte [@aws-sdk/client-s3](#) en la referencia de la API.

Llame a los servicios de forma asíncrona

Todas las solicitudes que se realizan a través del SDK son asíncronas. Es importante tener esto en cuenta al escribir scripts de navegador. JavaScript la ejecución en un navegador web normalmente tiene un único hilo de ejecución. Tras realizar una llamada asíncrona a un AWS servicio, el script del navegador sigue ejecutándose y, en ese proceso, puede intentar ejecutar código que dependa de ese resultado asíncrono antes de que regrese.

Hacer llamadas asíncronas a un AWS servicio incluye administrar esas llamadas para que el código no intente usar los datos antes de que estén disponibles. En los temas de esta sección se explica la necesidad de administrar llamadas asíncronas y técnicas diferentes de detalles que puede utilizar para administrarlas.

Aunque puede usar cualquiera de estas técnicas para gestionar las llamadas asíncronas, le recomendamos que utilice `async/await` para todos los códigos nuevos.

`async/await`

Le recomendamos que utilice esta técnica, ya que es el comportamiento predeterminado en la versión 3.

promesa

Utilice esta técnica en navegadores que no admitan `async/await`.

devolución de llamada

Evite el uso de callbacks excepto en casos muy sencillos. Sin embargo, puede que le resulte útil en escenarios de migración.

Temas

- [Administra las llamadas asíncronas](#)
- [Usa `async/await`](#)
- [Utilice JavaScript las promesas](#)
- [Usa una función de devolución de llamada anónima](#)

Administra las llamadas asíncronas

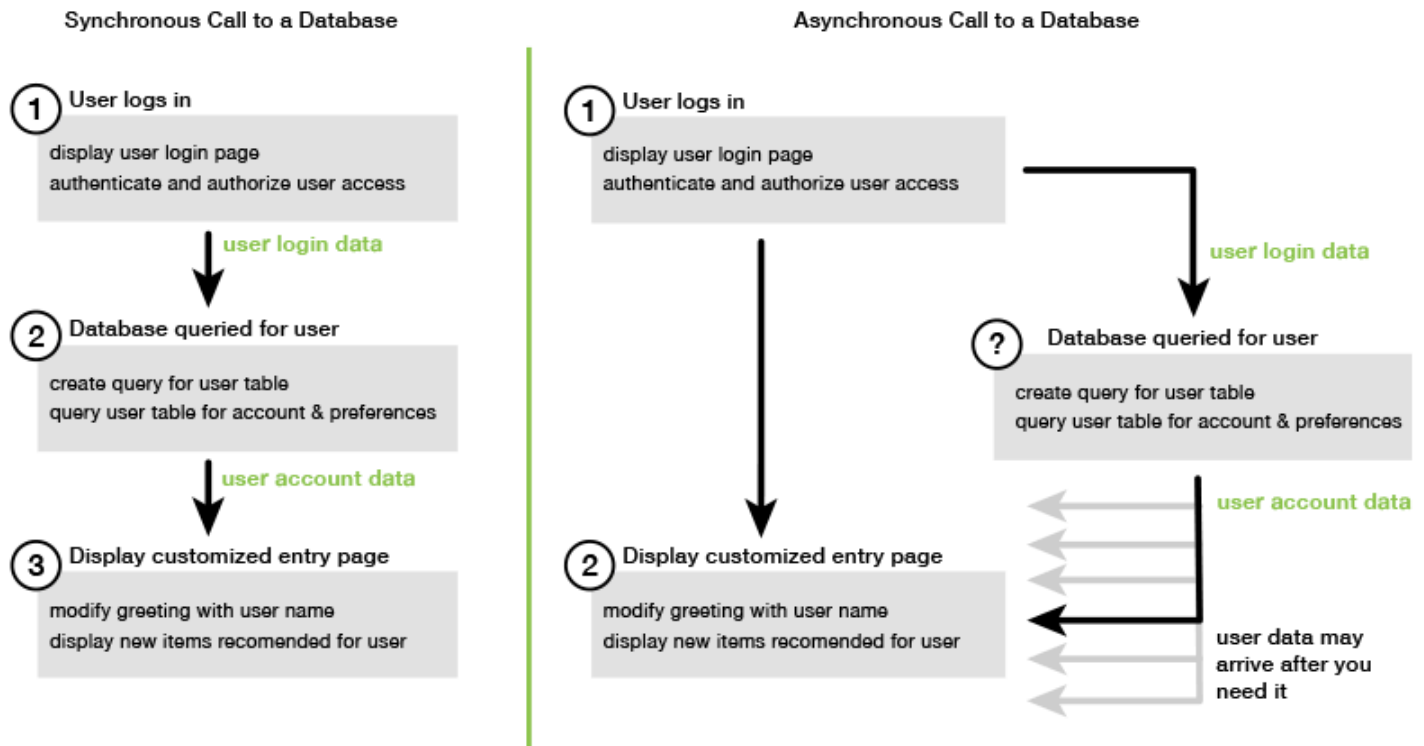
Por ejemplo, la página de inicio de un sitio web de e-commerce permite iniciar sesión a los clientes que regresan. Parte del beneficio para los clientes que inician sesión es que, después de iniciar sesión, el sitio se personaliza a sí mismo para adaptarse a sus preferencias concretas. Para que esto suceda:

1. El cliente debe iniciar sesión y haberse validado con sus credenciales de inicio de sesión.
2. Las preferencias del cliente se solicitan a partir de una base de datos del cliente.
3. La base de datos proporciona las preferencias del cliente que se utilizan para personalizar el sitio antes de que se cargue la página.

Si dichas tareas se ejecutan de forma síncrona, cada una tiene que finalizar antes de que la siguiente pueda comenzar. La página web no puede acabar de cargarse hasta que no lleguen las preferencias del cliente desde la base de datos. Sin embargo, después de que la consulta de la base de datos se

envíe al servidor, la recepción de los datos del cliente se puede retrasar o incluso generar errores debido a atascos en la red, un tráfico de base de datos excepcionalmente alto o una conexión de dispositivo móvil de mala calidad.

Para evitar que el sitio web se congele en estas condiciones, llame a la base de datos de forma asíncrona. Después de que se ejecute la llamada a la base de datos y se envíe su solicitud asíncrona, el código sigue ejecutándose según lo previsto. Si no administra correctamente la respuesta de una llamada asíncrona, el código puede intentar utilizar información que espera recibir de la base de datos cuando dichos datos todavía no están disponibles.



Usa async/await

En lugar de usar promesas, debería considerar la posibilidad de utilizar `async/await`. Las funciones asíncronas son más simples y requieren menos código reutilizable que las promesas. `await` solo se puede usar en las funciones asíncronas para esperar asíncronicamente un valor.

En el siguiente ejemplo, se usa `async/await` para enumerar todas las tablas de Amazon DynamoDB en `us-west-2`.

Note

Para ejecutar este ejemplo:

- Instale el cliente de AWS SDK for JavaScript DynamoDB npm `install @aws-sdk/client-dynamodb` ingresándolo en la línea de comandos de su proyecto.
- Asegúrese de haber configurado AWS las credenciales correctamente. Para obtener más información, consulte [Establece las credenciales](#).

```
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

No todos los navegadores admiten `async/await`. Consulte las [funciones asíncronas](#) para obtener una lista de navegadores compatibles con `async/await`.

Utilice JavaScript las promesas

Utilice el método AWS SDK for JavaScript v3 del cliente de servicio (`ListTablesCommand`) para realizar la llamada de servicio y gestionar el flujo asíncrono en lugar de utilizar devoluciones de llamada. El siguiente ejemplo muestra cómo obtener los nombres de sus tablas de Amazon DynamoDB en `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);
});
```

Coordina múltiples promesas

En algunas situaciones, el código debe realizar varias llamadas asíncronas que requieren acción solo cuando todas han tenido una devolución correcta. Si administra estas llamadas a métodos asíncronos individuales con promesas, puede crear una promesa adicional que utilice el método `all`.

Este método cumple esta promesa paraguas en el momento en que transfiere la matriz de promesas al método siempre y cuando dicha matriz de promesas se cumpla. Se transfiere a la función de devolución de llamada una matriz de los valores de las promesas que se transfieren al método `all`.

En el ejemplo siguiente, una AWS Lambda función debe realizar tres llamadas asíncronas a Amazon DynamoDB, pero solo puede completarlas después de cumplir las promesas de cada llamada.

```
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;
```

Compatibilidad del navegador y Node.js con las promesas

Support for native JavaScript promise (ECMAScript 2015) depende del JavaScript motor y la versión en la que se ejecute el código. Para ayudarte a determinar la compatibilidad con las JavaScript promesas en cada entorno en el que deba ejecutarse el código, consulta la tabla de compatibilidad de [ECMAScript](#) en GitHub

Usa una función de devolución de llamada anónima

Cada método de objeto de servicio puede aceptar una función de devolución de llamada anónima como último parámetro. La firma de dicha función de devolución de llamada es de la siguiente manera:

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

Esta función de devolución de llamada se ejecuta cuando se devuelve una respuesta correcta o datos de error. Si la llamada al método se realiza correctamente, el contenido de la respuesta está disponible en la función de devolución de llamada en el parámetro `data`. Si la llamada no se realiza correctamente, se proporcionan los detalles sobre el error en el parámetro `error`.

Normalmente el código contenido en la función de devolución de llamada realiza una prueba para detectar errores. Si el resultado de la prueba devuelve errores, los procesará. Si no se devuelve ningún error, el código recuperará los datos de la respuesta en el parámetro `data`. La forma básica de la función de devolución de llamada es similar a la de este ejemplo.

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

En el ejemplo anterior, los detalles del error o de los datos devueltos se registran en la consola. A continuación se muestra un ejemplo que muestra una función de devolución de llamada transferida como parte de una llamada a un método en un objeto de servicio.

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

Cree solicitudes de clientes de servicio

Hacer solicitudes a los clientes AWS de servicio es sencillo. La versión 3 (V3) del SDK JavaScript permite enviar solicitudes.

Note

También puede realizar operaciones con los comandos de la versión 2 (V2) cuando utilice la V3 del SDK para JavaScript. Para obtener más información, consulte [Uso de comandos de la versión 2](#).

Para enviar una solicitud:

1. Inicialice un objeto de cliente con la configuración deseada, como una región de AWS específica.
2. (Opcional) Cree un objeto JSON de solicitud con los valores de la solicitud, como el nombre de un bucket específico de Amazon S3. Puede examinar los parámetros de la solicitud consultando el tema de referencia de la API correspondiente a la interfaz con el nombre asociado al método del cliente. Por ejemplo, si utiliza el método *AbcCommand* cliente, la interfaz de solicitud es *AbcInput*.
3. Si lo desea, puede inicializar un comando de servicio con el objeto de solicitud como entrada.
4. Llame a `send` en el cliente con el objeto de comando como entrada.

Por ejemplo, para obtener una lista de sus tablas de Amazon DynamoDB en `us-west-2`, puede hacerlo con `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);
  }
})();
```

Gestione las respuestas de los clientes del servicio

Tras llamar a un método de cliente de servicio, devuelve una instancia de objeto de respuesta de una interfaz con el nombre asociado al método de cliente. Por ejemplo, si utiliza el método *AbcCommand* cliente, el objeto de respuesta es del tipo *AbcResponse*(interfaz).

Acceda a los datos devueltos en la respuesta

El objeto de respuesta contiene los datos, como propiedades, devueltos por la solicitud de servicio.

En [Cree solicitudes de clientes de servicio](#), el comando `ListTablesCommand` devolvió los nombres de las tablas de la propiedad `TableNames` de la respuesta.

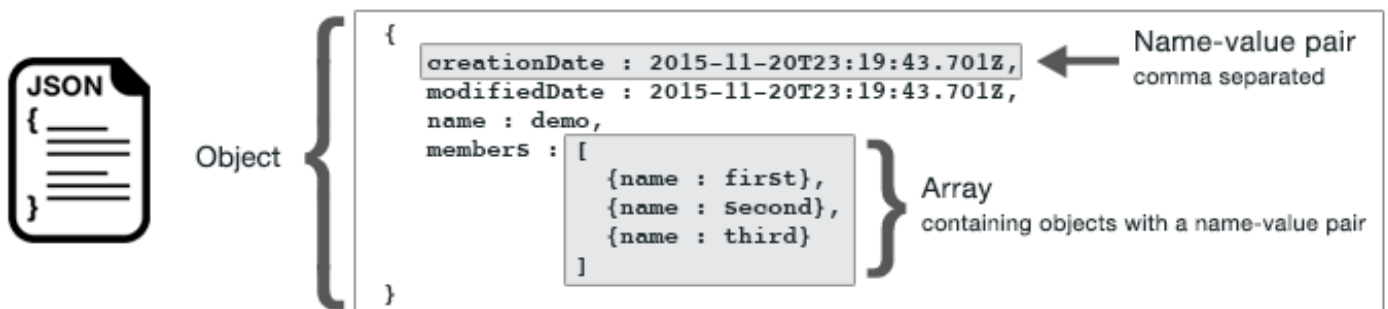
Acceder a la información sobre errores

Si se produce un error en un comando, se produce una excepción. Puede gestionar la excepción según lo necesite.

Trabaja con JSON

JSON es un formato para el intercambio de datos que pueden leer tanto humanos como las máquinas. Aunque el nombre JSON es un acrónimo de JavaScript Object Notation, el formato de JSON es independiente de cualquier lenguaje de programación.

AWS SDK for JavaScript Utiliza JSON para enviar datos a los objetos de servicio al realizar solicitudes y recibe los datos de los objetos de servicio en forma de JSON. Para obtener más información sobre JSON, consulte json.org.



JSON representa los datos de dos formas:

- Como un objeto, que es una colección sin ordenar de pares de nombre-valor. Un objeto se define entre las llaves izquierda (`{`) y derecha (`}`). Cada par de nombre-valor comienza por el nombre, seguido de dos puntos, seguido del valor. Los pares de nombre-valor están separados por comas.
- Como una matriz, que es una colección ordenada de valores. Una matriz se define entre los corchetes izquierdo (`[`) y derecho (`]`). Los elementos de la matriz están separados por comas.

A continuación, se muestra un ejemplo de un objeto JSON que contiene una matriz de objetos en la que los objetos representan las naipes de un juego de cartas. Cada carta está definida con dos pares de nombre-valor, uno que especifica un valor único para identificar la carta y otra que especifica una dirección URL que apunta a la imagen de la carta correspondiente.

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

Parámetros de JSON como objeto de servicio

A continuación se muestra un ejemplo de JSON sencillo que se utiliza para definir los parámetros de una llamada a un objeto de servicio de Lambda.

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

El objeto `params` se define mediante tres pares de nombre-valor, separados por comas e incluidos entre llaves (izquierda y derecha). Cuando se proporcionan parámetros a una llamada de método de objeto de servicio, los nombres se determinan mediante nombres de parámetros para el método de objeto de servicio al que tiene previsto llamar. Al invocar una función de Lambda, `FunctionName`, `Payload` y `LogType` son los parámetros que se utilizan para llamar al método `invoke` de un objeto de servicio de Lambda.

Cuando transfiera parámetros a una llamada de método de objeto de servicio, proporcione el objeto JSON a la llamada al método, tal y como se muestra en el siguiente ejemplo de invocación a una función de 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 ejemplos JavaScript de código

Los temas de esta sección contienen ejemplos de cómo utilizarlos AWS SDK for JavaScript con las API de varios servicios para llevar a cabo tareas comunes.

Encuentra el código fuente de estos y otros ejemplos en el [repositorio de ejemplos de AWS código en GitHub](#). Para proponer un nuevo ejemplo de código para que el equipo de AWS documentación considere producirlo, crea una solicitud. El equipo está buscando crear ejemplos de código que abarquen situaciones y casos de uso más amplios, en comparación con fragmentos de código sencillos que tratan solo llamadas a la API individuales. Para obtener instrucciones, consulte la sección sobre creación de código en las [directrices de contribución sobre GitHub](#).

Important

Estos ejemplos utilizan la sintaxis de importación y exportación de ECMAScript6.

- Esto requiere la versión 14.17 o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).
- Si prefiere utilizar la sintaxis CommonJS, consulte las pautas de conversión [Sintaxis ES6/commonJS de JavaScript](#).

Temas

- [Sintaxis ES6/commonJS de JavaScript](#)
- [Ejemplos de Amazon DynamoDB](#)
- [Ejemplos de AWS Elemental MediaConvert](#)
- [Ejemplos de Lambda](#)
- [Ejemplos de Amazon Lex](#)
- [Ejemplos de Amazon Polly](#)
- [Ejemplos de Amazon Redshift](#)
- [Ejemplos de Amazon Simple Email Service](#)
- [Ejemplos de Amazon Simple Notification Service](#)
- [Ejemplos de Amazon Transcribe](#)
- [Configuración de Node.js en una instancia de Amazon EC2](#)
- [Crear una aplicación para enviar datos a DynamoDB](#)
- [Crear una aplicación de transcripción con usuarios autenticados](#)
- [Invocación de Lambda con API Gateway](#)
- [Creación de flujos de trabajo de AWS sin servidor usando AWS SDK for JavaScript](#)
- [Crear eventos programados para ejecutar funciones de Lambda](#)
- [Creación de un chatbot de Amazon Lex](#)
- [Creación de una aplicación de mensajería de ejemplo](#)

Sintaxis ES6/commonJS de JavaScript

Los ejemplos de código de AWS SDK for JavaScript están escritos en ECMAScript 6 (ES6). ES6 incorpora una nueva sintaxis y nuevas funciones para que su código sea más moderno y legible, y haga más cosas.

ES6 requiere que utilice la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#). No obstante, si lo prefiere, puede convertir cualquiera de nuestros ejemplos a la sintaxis CommonJS. Para ello, siga las instrucciones que se indican a continuación:

- Elimine module de package.json del entorno de su proyecto.

- Convierta todas las declaraciones import de ES6 en declaraciones require de CommonJS. Por ejemplo, convierta:

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

A su equivalente en CommonJS:

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

- Convierta todas las declaraciones import de ES6 en declaraciones require de CommonJS. Por ejemplo, convierta:

```
export {s3}
```

A su equivalente en CommonJS:

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

El siguiente ejemplo muestra el ejemplo de código para crear un bucket de Amazon S3 tanto en ES6 como en 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

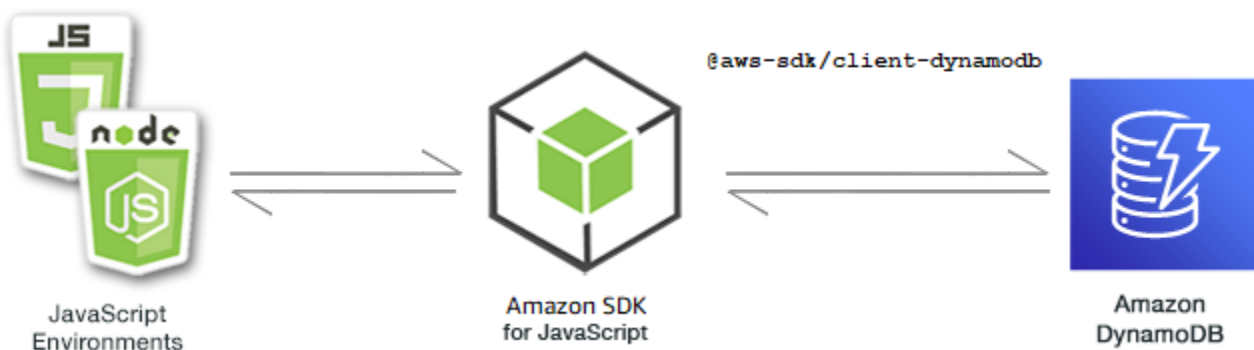
```
// Get service clients module and commands using CommonJS syntax.
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("../libs/s3Client.js");

// 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();
```

Ejemplos de Amazon DynamoDB

Amazon DynamoDB es una base de datos de la nube NoSQL completamente administrada compatible con modelos de almacenamiento de documentos y clave-valor. Puede crear tablas sin esquema para los datos sin necesidad de aprovisionar ni de mantener servidores de bases de datos dedicados.



La API de JavaScript para DynamoDB se expone a través de las clases de cliente `DynamoDB`, `DynamoDBStreams` y `DynamoDB.DocumentClient`. Para obtener más información acerca

de cómo utilizar las clases de cliente de DynamoDB, consulte [Clase: DynamoDB](#), [Clase: DynamoDBStreams](#) y [Clase: utilidad de DynamoDB](#) en la referencia de la API.

Temas

- [Creación y uso de tablas en DynamoDB](#)
- [Lectura y escritura de un único elemento en DynamoDB](#)
- [Lectura y escritura de elementos en lotes en DynamoDB](#)
- [Consulta y examen de una tabla de DynamoDB](#)
- [Uso del cliente de documentos de DynamoDB](#)

Creación y uso de tablas en DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo crear y administrar tablas utilizadas para almacenar y recuperar datos desde DynamoDB.

El escenario

Al igual que otros sistemas de base de datos, DynamoDB almacena datos en tablas. Una tabla de DynamoDB es una colección de datos que se organizan en elementos similares a las filas. Para almacenar u obtener acceso a datos de DynamoDB, debe crear y trabajar con tablas.

En este ejemplo, va a utilizar una serie de módulos Node.js para realizar operaciones básicas con una tabla de DynamoDB. El código utiliza el para crear y trabajar con tablas mediante los métodos siguientes de la clase de cliente de DynamoDB:

- [CreateTableCommand](#)
- [ListTablesCommand](#)
- [DescribeTableCommand](#)
- [DeleteTableCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node.js e instale los módulos AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Instale el SDK para el cliente DynamoDB de JavaScript. Para obtener más información, consulte [¿Qué novedades incluye la versión 3?](#).
- Cree un archivo de configuraciones compartido con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

Important

En estos ejemplos se utiliza ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

Sin embargo, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Note

Para obtener información sobre los tipos de datos utilizados en estos ejemplos, consulte [Tipos de datos y reglas de denominación compatibles en Amazon DynamoDB](#).

Creación de una tabla

Cree un módulo de Node.js con el nombre de archivo `create-table.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para crear una tabla, que en este ejemplo incluye el nombre y el tipo de datos para cada atributo, el esquema de claves, el nombre de la tabla y las unidades de rendimiento que deben aprovisionarse. Llame al método `CreateTableCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de sus tablas

Cree un módulo de Node.js con el nombre de archivo `list-tables.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para obtener una lista de las tablas, que en este ejemplo limita el número de tablas enumeradas a 10. Llame al método `ListTablesCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Descripción de una tabla

Cree un módulo de Node.js con el nombre de archivo `describe-table.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para describir un método `DescribeTableCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una tabla

Cree un módulo de Node.js con el nombre de archivo `delete-table.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para eliminar una tabla. En este ejemplo, el nombre de la tabla se proporciona como parámetro de la línea de comandos. Llame al método `DeleteTableCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Lectura y escritura de un único elemento en DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo añadir un elemento a una tabla de DynamoDB.
- Cómo recuperar un elemento de una tabla de DynamoDB.
- Cómo eliminar un elemento de una tabla de DynamoDB

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para leer y escribir un elemento en una tabla de DynamoDB mediante los métodos siguientes de la clase de cliente de DynamoDB:

- [PutItemCommand](#)
- [UpdateItemCommand](#)
- [GetItemCommand](#)
- [DeleteItemCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node.js e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartido con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.
- Cree una tabla de DynamoDB a cuyos elementos pueda obtener acceso. Para obtener más información acerca de cómo crear una tabla de DynamoDB consulte [Creación y uso de tablas en DynamoDB](#).

⚠ Important

En estos ejemplos se utiliza ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

Sin embargo, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

ℹ Note

Para obtener información sobre los tipos de datos utilizados en estos ejemplos, consulte [Tipos de datos y reglas de denominación compatibles en Amazon DynamoDB](#).

Escritura de un elemento

Cree un módulo de Node.js con el nombre de archivo `put-item.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para añadir un elemento, que en este ejemplo incluye el nombre de la tabla y un mapa que define los atributos que deben establecerse y los valores para cada atributo. Llame al método `PutItemCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Actualización de un elemento

Cree un módulo de Node.js con el nombre de archivo `update-item.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para añadir un elemento, que en este ejemplo incluye el nombre de la tabla, la clave a actualizar, la expresión de fecha y un mapa que define los nuevos nombres de atributo y los valores para cada atributo. Llame al método `UpdateItemCommand` del objeto de servicio de 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": { B00L: "false" },
    },
  },
```



```
    ReturnValues: "ALL_NEW",
  });

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

Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de un elemento

Cree un módulo de Node.js con el nombre de archivo `get-item.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Para identificar el elemento que se desea obtener, debe proporcionar el valor de la clave principal de ese elemento en la tabla. De forma predeterminada, el método `GetItemCommand` devuelve todos los valores de atributos definidos para el elemento. Para obtener únicamente un subconjunto de todos los posibles valores de atributo, especifique una expresión de proyección.

Cree un objeto JSON que contenga los parámetros necesarios para obtener un elemento, que en este ejemplo incluye el nombre de la tabla, el nombre y el valor de la clave para el elemento que recibe y una expresión de proyección que identifica el atributo del elemento que desea recuperar. Llame al método `GetItemCommand` del objeto de servicio de DynamoDB.

El siguiente ejemplo de código recupera un elemento de una tabla con una clave principal compuesta únicamente por una clave de partición y no por una clave de partición ni por una clave de clasificación. Si la tabla tiene una clave principal compuesta por una clave de partición y una clave de clasificación, también tiene que especificar el nombre y el atributo de la clave de clasificación.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un elemento

Cree un módulo de Node.js con el nombre de archivo `delete-item.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para eliminar un elemento, que, en este ejemplo, contiene el nombre de la tabla y tanto el nombre como el valor de la clave del elemento que va a eliminar. Llame al método `DeleteItemCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Lectura y escritura de elementos en lotes en DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo leer y escribir lotes de elementos en una tabla de DynamoDB.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para poner un lote de elementos de una tabla de DynamoDB y leer un lote de elementos. El código utiliza el SDK para JavaScript para realizar operaciones de lectura y escritura por lotes mediante los métodos siguientes de clase de cliente de DynamoDB:

- [BatchGetItemCommand](#)
- [BatchWriteItemCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartido con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.
- Cree una tabla de DynamoDB a cuyos elementos pueda obtener acceso. Para obtener más información acerca de cómo crear una tabla de DynamoDB consulte [Creación y uso de tablas en DynamoDB](#).

Important

En estos ejemplos se utiliza ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

Sin embargo, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Note

Para obtener información sobre los tipos de datos utilizados en estos ejemplos, consulte [Tipos de datos y reglas de denominación compatibles en Amazon DynamoDB](#).

Lectura de elementos en lotes

Cree un módulo de Node.js con el nombre de archivo `batch-get-item.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para obtener un lote de elementos, que en este ejemplo incluye el nombre de una o varias tablas donde se leerá, los valores de claves para leer en cada tabla y la expresión de proyección que especifica los atributos que se devolverán. Llame al método `BatchGetItemCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Escritura de elementos en lotes

Cree un módulo de Node.js con el nombre de archivo `batch-write-item.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para obtener un lote de elementos, que en este ejemplo incluye la tabla en la que desea escribir elementos, las claves que desea escribir para cada elemento y los atributos junto con sus valores. Llame al método `BatchWriteItemCommand` del objeto de servicio de 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"] },
            },
          },
        },
      ],
    },
  });
}
```

```
        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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Consulta y examen de una tabla de DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo consultar y examinar una tabla de DynamoDB para buscar elementos.

El escenario

Con las consultas se encuentran elementos de una tabla o un índice secundario usando únicamente valores de atributos de la clave principal. Debe proporcionar un nombre de clave de partición y un valor que deben buscarse. También puede proporcionar un nombre y un valor de clave de ordenación y utilizar un operador de comparación para ajustar los resultados de búsqueda. El examen busca elementos comprobando cada uno de ellos en la tabla especificada.

En este ejemplo, va a utilizar una serie de módulos Node.js para identificar uno o varios elementos que desea recuperar de una tabla de DynamoDB . El código utiliza el SDK para JavaScript para consultar y analizar tablas mediante los métodos siguientes de la clase de cliente de DynamoDB:

- [QueryCommand](#)
- [ScanCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node.js e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartido con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.
- Cree una tabla de DynamoDB a cuyos elementos pueda obtener acceso. Para obtener más información acerca de cómo crear una tabla de DynamoDB consulte [Creación y uso de tablas en DynamoDB](#).

Important

En estos ejemplos se utiliza ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

Sin embargo, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Note

Para obtener información sobre los tipos de datos utilizados en estos ejemplos, consulte [Tipos de datos y reglas de denominación compatibles en Amazon DynamoDB](#).

Consulta de una tabla

Cree un módulo de Node.js con el nombre de archivo `query.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para consultar la tabla, que en este ejemplo incluye el nombre de la tabla, los `ExpressionAttributeValues` necesarios para la consulta, una `KeyConditionExpression` que utilice dichos valores para definir qué elementos devuelve la consulta, así como los nombres de los valores de atributos que deben devolverse para cada elemento. Llame al método `QueryCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node query.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Análisis de una tabla

Cree un módulo de Node.js con el nombre de archivo `scan.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Para acceder a DynamoDB, cree un objeto de servicio de cliente de DynamoDB. Cree un objeto JSON que contenga los parámetros necesarios para examinar la tabla en búsqueda de elementos, que en este ejemplo incluyen el nombre de la tabla, la lista de los valores de atributos que deben devolverse por cada elemento coincidente y una expresión para filtrar el conjunto de resultados para encontrar elementos que contengan una frase especificada. Llame al método `ScanCommand` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node scan.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso del cliente de documentos de DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo acceder a una tabla de DynamoDB utilizando las utilidades de DynamoDB.

El escenario

El cliente de documentos de DynamoDB simplifica el trabajo con los elementos ya que abstrae la noción de valores de atributos. Esta abstracción comenta los tipos de JavaScript nativos suministrados como parámetros de entrada y convierte datos de respuesta comentados en tipos de JavaScript nativos.

Para obtener más información sobre el cliente de documentos de DynamoDB, consulte [@aws-sdk/lib-dynamodb README](#) en GitHub. Para obtener más información sobre la programación con Amazon DynamoDB, consulte [Programación con DynamoDB](#) en la Guía para desarrolladores de Amazon DynamoDB.

En este ejemplo, va a utilizar una serie de módulos Node.js para realizar operaciones básicas en una tabla de DynamoDB usando utilidades de DynamoDB. El código utiliza el SDK para JavaScript para consultar y analizar tablas mediante los métodos siguientes de la clase de cliente de documentos de DynamoDB:

- [GetCommand](#)
- [PutCommand](#)
- [UpdateCommand](#)
- [QueryCommand](#)
- [DeleteCommand](#)

Para obtener más información sobre la configuración del cliente de documentos de DynamoDB, consulte [@aws-sdk/lib-dynamodb](#).

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node.js e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartido con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.
- Cree una tabla de DynamoDB a cuyos elementos pueda obtener acceso. Para obtener más información acerca de cómo crear una tabla de DynamoDB utilizando el SDK para JavaScript, consulte [Creación y uso de tablas en DynamoDB](#). También puede utilizar la [consola de DynamoDB](#) para crear una tabla.

Important

En estos ejemplos se utiliza ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

Sin embargo, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Note

Para obtener información sobre los tipos de datos utilizados en estos ejemplos, consulte [Tipos de datos y reglas de denominación compatibles en Amazon DynamoDB](#).

Obtención de un elemento de una tabla

Cree un módulo de Node.js con el nombre de archivo `get.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios. Esto incluye `@aws-sdk/lib-dynamodb`, un paquete de bibliotecas que proporciona la funcionalidad

de cliente de documentos a `@aws-sdk/client-dynamodb`. A continuación, establezca la configuración que se muestra a continuación para ordenar y desordenar (como segundo parámetro opcional) durante la creación del cliente de documentos. A continuación, cree los clientes. Ahora cree un objeto JSON que contenga los parámetros necesarios para obtener un elemento de la tabla, que en este ejemplo incluye el nombre de la tabla, el nombre de la clave hash en dicha tabla y el valor de la clave hash para el elemento que desea obtener. Llame al método `GetCommand` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node get.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Colocación de un elemento en una tabla

Cree un módulo de Node.js con el nombre de archivo `put.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios. Esto incluye `@aws-sdk/lib-dynamodb`, un paquete de bibliotecas que proporciona la funcionalidad de cliente de documentos a `@aws-sdk/client-dynamodb`. A continuación, establezca la configuración que se muestra a continuación para ordenar y desordenar (como segundo parámetro opcional) durante la creación del cliente de documentos. A continuación, cree los clientes. Cree un

objeto JSON que contenga los parámetros necesarios para escribir un elemento en la tabla, que en este ejemplo incluye el nombre de la tabla y una descripción del elemento para añadir o actualizar que incluye el clave hash y el valor, y los nombres y los valores de atributos que se van a establecer en el elemento. Llame al método `PutCommand` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node put.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Actualización de un elemento en una tabla

Cree un módulo de Node.js con el nombre de archivo `update.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios. Esto incluye `@aws-sdk/lib-dynamodb`, un paquete de bibliotecas que proporciona la funcionalidad de cliente de documentos a `@aws-sdk/client-dynamodb`. A continuación, establezca la configuración que se muestra a continuación para ordenar y desordenar (como segundo parámetro opcional) durante la creación del cliente de documentos. A continuación, cree los clientes. Cree un objeto JSON que contenga los parámetros necesarios para escribir un elemento en la tabla, que en este ejemplo incluye el nombre de la tabla, la clave del elemento que se va a actualizar, un conjunto de `UpdateExpressions` que definen los atributos del elemento que se va a actualizar con tokens

a los que asigna valores en los parámetros `ExpressionAttributeValues`. Llame al método `UpdateCommand` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node update.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Consulta de una tabla

Cree un módulo de Node.js con el nombre de archivo `query.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios. Esto incluye `@aws-sdk/lib-dynamodb`, un paquete de bibliotecas que proporciona la funcionalidad de cliente de documentos a `@aws-sdk/client-dynamodb`. Cree un objeto JSON que contenga los parámetros necesarios para consultar la tabla, que en este ejemplo incluye el nombre de la tabla, los `ExpressionAttributeValues` necesarios para la consulta y una `KeyConditionExpression`

que utilice dichos valores para definir qué elementos devuelve la consulta. Llame al método `QueryCommand` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node query.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un elemento de una tabla

Cree un módulo de Node.js con el nombre de archivo `delete.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios. Esto incluye `@aws-sdk/lib-dynamodb`, un paquete de bibliotecas que proporciona la funcionalidad de cliente de documentos a `@aws-sdk/client-dynamodb`. A continuación, establezca la configuración que se muestra a continuación para ordenar y desordenar (como segundo parámetro opcional) durante la creación del cliente de documentos. A continuación, cree los clientes. Para acceder a DynamoDB, cree un objeto de DynamoDB. Cree un objeto JSON que contenga los

parámetros necesarios para eliminar un elemento de la tabla, que en este ejemplo incluye el nombre de la tabla, y el nombre y el valor de la clave hash del elemento que desea eliminar. Llame al método `DeleteCommand` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node delete.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de AWS Elemental MediaConvert

AWS Elemental MediaConvert es un servicio de transcodificación de vídeo basado en archivos con características apropiadas para los medios de difusión. Puede utilizarlo para crear recursos para su difusión y para la entrega de vídeo bajo demanda (VOD) a través de Internet. Para obtener más información, consulte la [Guía del usuario de AWS Elemental MediaConvert](#).

La API de JavaScript para MediaConvert se expone a través de la clase de cliente de MediaConvert. Para obtener más información, consulte [Clase: MediaConvert](#) en la referencia de la API.

Temas

- [Obtener el punto de conexión específico de su región para MediaConvert](#)
- [Creación y administración de trabajos de transcodificación en MediaConvert](#)
- [Uso de plantillas de trabajo en MediaConvert](#)

Obtener el punto de conexión específico de su región para MediaConvert



Este ejemplo de código de Node.js muestra:

- Cómo recuperar el punto de de conexión específico de la región de MediaConvert.

El escenario

En este ejemplo, se utiliza un módulo de Node.js para llamar a MediaConvert y recuperar el punto de conexión específico de la región. Puede recuperar la URL del punto de conexión desde el punto de conexión predeterminado del servicio, por lo que todavía no necesita el punto de conexión específico de la región. El código utiliza el SDK para JavaScript para recuperar este punto de conexión mediante el método siguiente de la clase de cliente de MediaConvert:

- [DescribeEndpointsCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.
- Cree un rol de IAM que conceda a MediaConvert acceso a los archivos de entrada y a los buckets de Amazon S3 donde se almacenan los archivos de salida. Para obtener más información, consulte [Configurar permisos de IAM](#) en la Guía del usuario de AWS Elemental MediaConvert.

⚠ Important

Este ejemplo usa ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js.

Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

Sin embargo, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Obtención de la URL del punto de conexión

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `emcClientGet.js`. Copie y pegue el siguiente código en él, lo que crea el objeto de cliente MediaConvert. Sustituya **REGION** por su región de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `emc_getendpoint.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para pasar los parámetros de solicitud vacíos para el método `DescribeEndpointsCommand` de la clase de cliente de MediaConvert. A continuación, llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_getendpoint.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación y administración de trabajos de transcodificación en MediaConvert



Este ejemplo de código de Node.js muestra:

- Cómo especificar el punto de conexión específico de la región para usarlo con MediaConvert.
- Cómo crear trabajos de transcodificación en MediaConvert.
- Cómo cancelar un trabajo de transcodificación.
- Cómo recuperar el JSON de un trabajo de transcodificación finalizado.
- Cómo recuperar una matriz JSON para un máximo de 20 de los últimos trabajos creados.

El escenario

En este ejemplo, se utiliza un módulo de Node.js para llamar a MediaConvert para crear y administrar trabajos de transcodificación. El código usa el SDK de JavaScript para realizar esta operación mediante los métodos de la clase de cliente de MediaConvert siguientes:

- [CreateJobCommand](#)

- [CancelJobCommand](#)
- [GetJobCommand](#)
- [ListJobsCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.
- Crear y configurar buckets de Amazon S3 que proporcionan almacenamiento para archivos de entrada de trabajo y archivos de salida. Para obtener información detallada, consulte [Crear almacenamiento para archivos](#) en la Guía del usuario de AWS Elemental MediaConvert.
- Cargar el vídeo de entrada en el bucket de Amazon S3 que ha provisionado para el almacenamiento de entrada. Para obtener una lista de los códecs y contenedores compatibles con la entrada de vídeo, consulte [Códecs y contenedores de entrada compatibles](#) en la Guía del usuario de AWS Elemental MediaConvert.
- Cree un rol de IAM que conceda a MediaConvert acceso a los archivos de entrada y a los buckets de Amazon S3 donde se almacenan los archivos de salida. Para obtener más información, consulte [Configurar permisos de IAM](#) en la Guía del usuario de AWS Elemental MediaConvert.

Important

Este ejemplo usa ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

Sin embargo, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Configuración del SDK

Configure el SDK como se mostró anteriormente, incluida la descarga de los clientes y paquetes necesarios. Como MediaConvert utiliza puntos de conexión personalizados para cada cuenta, también tiene que configurar la clase de cliente de MediaConvert para utilizar el punto de conexión específico de su región. Para ello, establezca el parámetro `endpoint` en `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";
```

Definición de un trabajo de transcodificación sencillo

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `emcClient.js`. Copie y pegue el siguiente código en él, lo que crea el objeto de cliente MediaConvert. Sustituya **REGION** por su región de AWS. Sustituya **ENDPOINT** por el punto de conexión de su cuenta de MediaConvert, que puede hacerlo en la página Cuenta de la consola de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `emc_createjob.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios. Cree el JSON que define los parámetros del trabajo de transcodificación.

Estos parámetros son bastante detallados. Puede utilizar la [AWS Elemental MediaConvertconsola](#) para generar los parámetros JSON del trabajo seleccionando la configuración del trabajo en la consola y, a continuación, seleccionando Mostrar JSON del trabajo en la parte inferior de la sección Trabajo. En este ejemplo, se muestra el JSON de un trabajo sencillo.

Note

Sustituya `JOB_QUEUE_ARN` por la cola de trabajos de MediaConvert, `IAM_ROLE_ARN` por el Nombre de recurso de Amazon (ARN) del rol de IAM, `OUTPUT_BUCKET_NAME` por el nombre del bucket de destino (por ejemplo, `"s3://OUTPUT_BUCKET_NAME/"`), y `INPUT_BUCKET_AND_FILENAME` por el bucket de entrada y el nombre de archivo (por ejemplo, `"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",
},
},
};
```

Creación de un trabajo de transcodificación

Después de crear el JSON de los parámetros del trabajo, llame al método `run` asíncrono para que invoque un objeto de servicio de cliente de `MediaConvert` mediante la transferencia de los parámetros. El ID del trabajo creado se devuelve en la respuesta `data`.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_createjob.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cancelación de un trabajo de transcodificación

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `emcClient.js`. Copie y pegue el siguiente código en él, lo que crea el objeto de cliente `MediaConvert`. Sustituya **REGION** por su región de AWS. Sustituya **ENDPOINT** por el punto de conexión de su cuenta de `MediaConvert`, que puede hacerlo en la página Cuenta de la consola de `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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `emc_canceljob.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la descarga de los clientes y paquetes necesarios. Cree el JSON que incluye el ID del trabajo que desea cancelar. Luego llame al método `CancelJobCommand` creando una promesa para invocar un objeto de servicio de cliente de MediaConvert mediante la transferencia de los parámetros. Gestione la respuesta en la devolución de llamada de la promesa.

Note

Sustituya *JOB_ID* por el ID del trabajo que desea cancelar.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_canceljob.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Listado de los trabajos de transcodificación recientes

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `emcClient.js`. Copie y pegue el siguiente código en él, lo que crea el objeto de cliente MediaConvert. Sustituya **REGION** por su región de AWS. Sustituya **ENDPOINT** por el punto de conexión de su cuenta de MediaConvert, que puede hacerlo en la página Cuenta de la consola de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `emc_listjobs.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree el JSON de los parámetros, incluidos los valores que especificarán si se debe ordenar la lista en orden ASCENDING o DESCENDING, el Nombre de recurso de Amazon (ARN) de la cola de trabajos que se va a comprobar y el estado de los trabajos que se deben incluir. Luego llame al método `ListJobsCommand` creando una promesa para invocar un objeto de servicio de cliente de MediaConvert mediante la transferencia de los parámetros.

Note

Sustituya **QUEUE_ARN** por el Nombre de recurso de Amazon (ARN) de la cola de trabajos que va a comprobar y **STATUS** por el estado de la cola.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_listjobs.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de plantillas de trabajo en MediaConvert



Este ejemplo de código de Node.js muestra:

- Cómo crear plantillas de trabajos de AWS Elemental MediaConvert.
- Cómo utilizar una plantilla de trabajo para crear un trabajo de transcodificación.
- Cómo generar una lista de todas sus plantillas de trabajo.

- Cómo eliminar plantillas de trabajos.

El escenario

El JSON necesario para crear un trabajo de transcodificación en MediaConvert es detallado, ya que contiene un gran número de opciones de configuración. Puede simplificar en gran medida la creación del trabajo guardando la configuración de funcionalidad comprobada en una plantilla de trabajo que pueda utilizar para crear trabajos posteriores. En este ejemplo, se utiliza un módulo de Node.js para llamar a MediaConvert para crear, utilizar y administrar plantillas de trabajos. El código usa el SDK de JavaScript para realizar esta operación mediante los métodos de la clase de cliente de MediaConvert siguientes:

- [CreateJobTemplateCommand](#)
- [CreateJobCommand](#)
- [DeleteJobTemplateCommand](#)
- [ListJobTemplatesCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.
- Cree un rol de IAM que conceda a MediaConvert acceso a los archivos de entrada y a los buckets de Amazon S3 donde se almacenan los archivos de salida. Para obtener más información, consulte [Configurar permisos de IAM](#) en la Guía del usuario de AWS Elemental MediaConvert.

Important

Estos ejemplos utilizan ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

Sin embargo, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Creación de una plantilla de trabajo

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `emcClient.js`. Copie y pegue el siguiente código en él, lo que crea el objeto de cliente MediaConvert. Sustituya **REGION** por su región de AWS. Sustituya **ENDPOINT** por el punto de conexión de su cuenta de MediaConvert, que puede hacerlo en la página Cuenta de la consola de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `emc_create_jobtemplate.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Especifique los parámetros JSON para la creación de plantillas. Puede utilizar la mayoría de los parámetros JSON de un trabajo anterior realizado correctamente para especificar los valores de Settings en la plantilla. En este ejemplo se utiliza la configuración de trabajo de [Creación y administración de trabajos de transcodificación en MediaConvert](#).

Llame al método `CreateJobTemplateCommand` creando una promesa para invocar un objeto de servicio de cliente de MediaConvert mediante la transferencia de los parámetros.

Note

Sustituya **JOB_QUEUE_ARN** por el Nombre de recurso de Amazon (ARN) de la cola de trabajos que desea comprobar y **BUCKET_NAME** por el nombre del bucket de Amazon S3 de destino (por ejemplo, "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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_create_jobtemplate.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de un trabajo de transcodificación a partir de una plantilla de trabajo

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `emcClient.js`. Copie y pegue el siguiente código en él, lo que crea el objeto de cliente MediaConvert. Sustituya **REGION** por su región de AWS. Sustituya **ENDPOINT** por el punto de conexión de su cuenta de MediaConvert, que puede hacerlo en la página Cuenta de la consola de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `emc_template_createjob.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree los parámetros JSON de creación del trabajo, como el nombre de la plantilla de trabajo que desea utilizar y el método Settings que se utilizará, que son específicos del trabajo que está creando. Luego llame al método CreateJobsCommand creando una promesa para invocar un objeto de servicio de cliente de MediaConvert mediante la transferencia de los parámetros.

Note

Sustituya JOB_QUEUE_ARN por el Nombre de recurso de Amazon (ARN) de la cola de trabajos que desea comprobar, KEY_PAIR_NAME por, TEMPLATE_NAME por, ROLE_ARN por el Nombre de recurso de Amazon (ARN) del rol y INPUT_BUCKET_AND_FILENAME por el bucket de entrada y el nombre del archivo (por ejemplo, "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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_template_createjob.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Generación de una lista de sus plantillas de trabajo

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `emcClient.js`. Copie y pegue el siguiente código en él, lo que crea el objeto de cliente MediaConvert. Sustituya **REGION** por su región de AWS. Sustituya **ENDPOINT** por el punto de conexión de su cuenta de MediaConvert, que puede hacerlo en la página Cuenta de la consola de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `emc_listtemplates.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para pasar los parámetros de solicitud vacíos para el método `listTemplates` de la clase de cliente `MediaConvert`. Incluya valores para determinar qué plantillas incluirá en la lista (`NAME`, `CREATION DATE`, `SYSTEM`), la cantidad que se incluirán en la lista y su orden de clasificación. Para llamar al método `ListTemplatesCommand`, cree una promesa para invocar un objeto de servicio de cliente de `MediaConvert` mediante la transferencia de los parámetros.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_listtemplates.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una plantilla de trabajo

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `emcClient.js`. Copie y pegue el siguiente código en él, lo que crea el objeto de cliente MediaConvert. Sustituya **REGION** por su región de AWS. Sustituya **ENDPOINT** por el punto de conexión de su cuenta de MediaConvert, que puede hacerlo en la página Cuenta de la consola de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `emc_deletetemplate.js`. Asegúrese de configurar el SDK como se ha mostrado anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir el nombre de la plantilla de trabajo que desea eliminar como parámetros para el método `DeleteJobTemplateCommand` de la clase de cliente MediaConvert. Para llamar al método `DeleteJobTemplateCommand`, cree una promesa para invocar un objeto de servicio de cliente de MediaConvert mediante la transferencia de los parámetros.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_deletetemplate.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de Lambda

Lambda es un servicio de computación sin servidor que le permite ejecutar código sin aprovisionar ni administrar servidores, crear una lógica de escalado de clústeres compatible con las cargas de trabajo, mantener integraciones de eventos ni administrar tiempos de ejecución.

La API de JavaScript para Lambda se expone a través de la clase de cliente de [LambdaService](#).

A continuación se muestra una lista de ejemplos que muestran cómo crear y usar funciones de Lambda con la v3 de AWS SDK for JavaScript:

- [Invocación de Lambda con API Gateway](#)
- [Crear eventos programados para ejecutar funciones de Lambda](#)

Ejemplos de Amazon Lex

Amazon Lex es un servicio de AWS para crear interfaces de conversación en aplicaciones utilizando voz y texto.

La API de JavaScript para Amazon Lex se expone a través de la clase de cliente [Lex Runtime Service](#).

- [Creación de un chatbot de Amazon Lex](#)

Ejemplos de Amazon Polly



Este ejemplo de código de Node.js muestra:

- Cargue el audio grabado con Amazon Polly a Amazon S3

El escenario

En este ejemplo, se utilizan una serie de módulos de Node.js para cargar automáticamente el audio grabado con Amazon Polly a Amazon S3 mediante estos métodos de la clase de cliente de Amazon S3:

- [StartSpeechSynthesisTaskCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure un entorno de proyecto para ejecutar JavaScript ejemplos de Node siguiendo las instrucciones de [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia las herramientas y los SDK de AWS.
- Cree una encuesta de roles de usuario de Amazon Cognito AWS Identity and Access Management (IAM) no autenticada: permisos SynthesizeSpeech y un grupo de identidades de Amazon Cognito con la función de IAM asociada. En la siguiente sección [Cree los recursos mediante el AWSAWS CloudFormation](#), se describe cómo crear estos recursos.

Note

En este ejemplo se utiliza Amazon Cognito, pero si no utiliza Amazon Cognito, el usuario debe cumplir con la siguiente política AWS de permisos de IAM

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

Cree los recursos mediante el AWS CloudFormation

AWS CloudFormation le permite crear y aprovisionar despliegues de AWS infraestructura de forma predecible y repetitiva. Para obtener más información al respecto AWS CloudFormation, consulte la Guía del [AWS CloudFormation usuario](#).

Para crear la AWS CloudFormation pila:

1. Instale y configure las AWS CLI siguientes instrucciones de la [Guía del AWS CLI usuario](#).
2. Cree un archivo con un nombre `setup.yaml` en el directorio raíz de la carpeta de su proyecto y copie [el contenido GitHub en él](#).

Note

La AWS CloudFormation plantilla se generó utilizando lo que AWS CDK está [disponible aquí GitHub](#). Para obtener más información sobre el AWS CDK, consulte la [Guía para AWS Cloud Development Kit \(AWS CDK\) desarrolladores](#).

3. Ejecute el siguiente comando desde la línea de comandos y reemplace `STACK_NAME` por un nombre único para la pila.

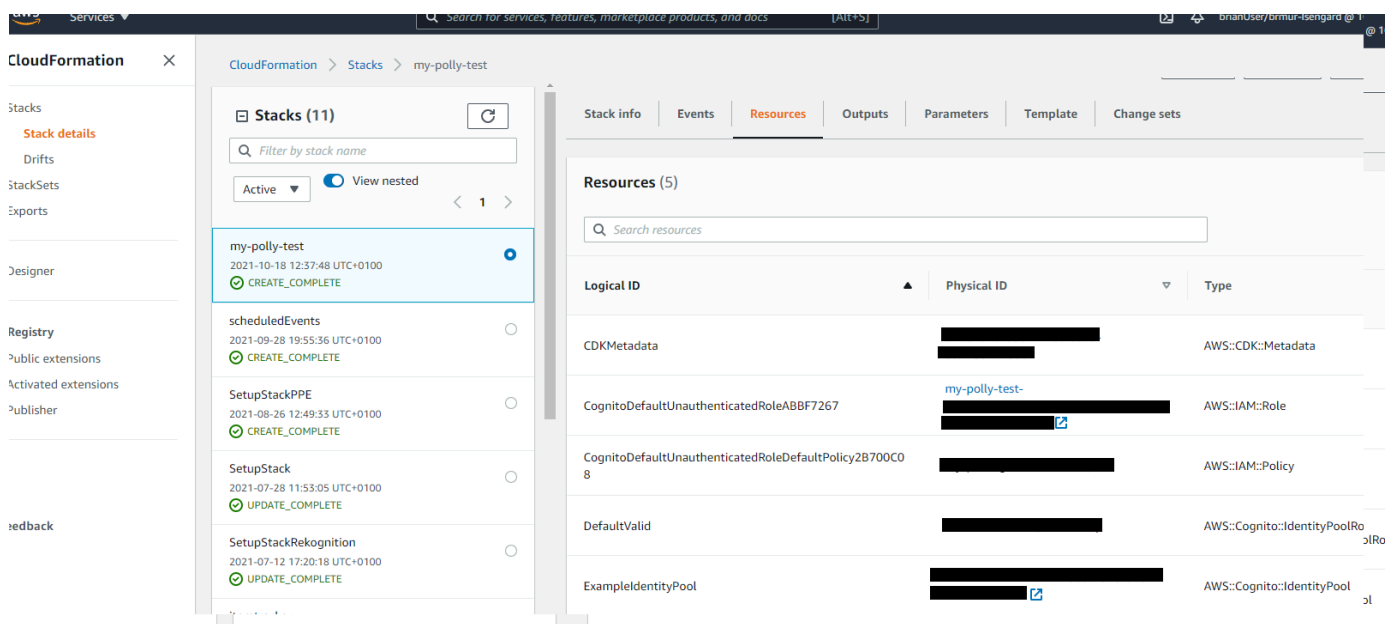
⚠ Important

El nombre de la pila debe ser único en AWS la región y en la AWS cuenta. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

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

Para obtener más información sobre los parámetros de los comandos `create-stack`, consulte la [Guía de referencia de comandos de AWS CLI](#) y la [Guía del usuario de AWS CloudFormation](#).

4. Ve a la consola AWS CloudFormation de administración, selecciona Pilas, elige el nombre de la pila y selecciona la pestaña Recursos para ver una lista de los recursos creados.



Cargue el audio grabado con Amazon Polly a Amazon S3

Cree un módulo de Node.js con el nombre de archivo `polly_synthesize_to_s3.js`. Asegúrese de configurar el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios. En el código, introduce la `REGION` y el `BUCKET_NAME`. Para acceder a Amazon Polly, cree un objeto de servicio al cliente de Polly. Sustituya `"IDENTITY_POOL_ID"` por el

IdentityPoolId de la página de muestra del grupo de identidades de Amazon Cognito que creó para este ejemplo. Esto también se transfiere a cada objeto de cliente.

Llame al método StartSpeechSynthesisCommand del objeto de servicio al cliente de Amazon Polly, sintetice el mensaje de voz y cárguelo en el bucket de 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",
};

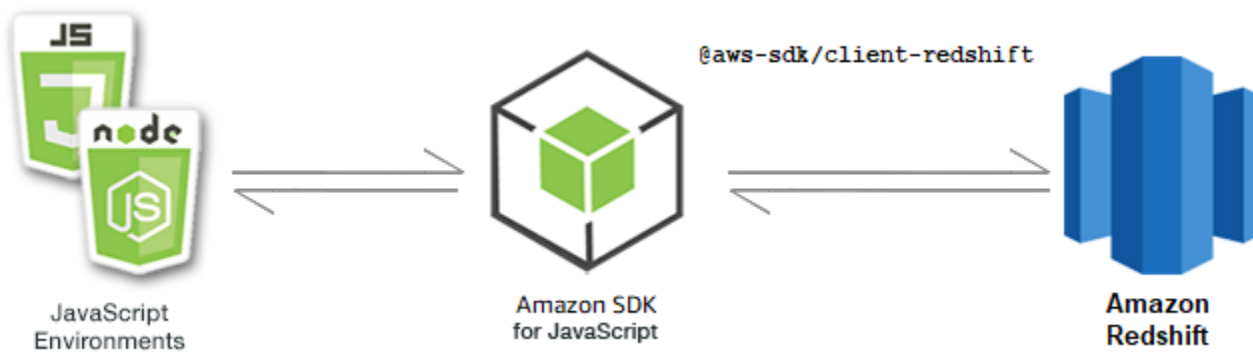
const run = async () => {
  try {
    await pollyClient.send(new StartSpeechSynthesisTaskCommand(params));
    console.log("Success, audio file added to " + params.OutputS3BucketName);
  } catch (err) {
    console.log("Error putting object", err);
  }
};

run();
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Ejemplos de Amazon Redshift

Amazon Redshift es un servicio de data warehouse administrado de varios petabytes en la nube. Un almacenamiento de datos de Amazon Redshift es una colección de recursos informáticos denominados nodos que se organizan en un grupo llamado clúster. Cada clúster ejecuta un motor Amazon Redshift y contiene una o más bases de datos.



La API de JavaScript para Amazon Redshift se expone a través de la clase de cliente de [Amazon Redshift](#).

Temas

- [Ejemplos de Amazon Redshift](#)

Ejemplos de Amazon Redshift

En este ejemplo, se utilizan una serie de módulos de Node.js para crear, modificar, describir los parámetros y, a continuación, eliminar clústeres de Amazon Redshift mediante los siguientes métodos de la clase de cliente de Redshift:

- [CreateClusterCommand](#)
- [ModifyClusterCommand](#)
- [DescribeClustersCommand](#)
- [DeleteClusterCommand](#)

Para obtener más información sobre los usuarios de Amazon Redshift, consulte la [Guía de introducción de Amazon Redshift](#).

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte

[Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de los SDK y las herramientas de AWS.

⚠ Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js.](#)
- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#)

Creará un clúster de Amazon Redshift.

En este ejemplo, se muestra cómo crear un clúster de Amazon Redshift con AWS SDK for JavaScript. Para obtener más información, consulte [CreateCluster](#).

⚠ Important

El clúster que está a punto de lanzarse se ejecutará en un entorno real (no en un entorno de pruebas). Debe pagar las tarifas de uso estándar de Amazon Redshift por el clúster hasta que lo elimine. Si elimina el clúster en la misma sesión en que lo creó, los cargos totales son mínimos.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `redshiftClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon Redshift. Sustituya **REGION** por su región de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `redshift-create-cluster.js`. Asegúrese de configurar el SDK como se indicó anteriormente, incluida la instalación de los clientes y paquetes necesarios. Cree un objeto de parámetros, especificando el tipo de nodo que se va a aprovisionar y las credenciales maestras de inicio de sesión de la instancia de base de datos creada automáticamente en el clúster y, por último, el tipo de clúster.

Note

Sustituya `CLUSTER_NAME` por el nombre de su clúster. En `NODE_TYPE`, especifique el tipo de nodo que se va a aprovisionar, como “dc2.large”, por ejemplo. `MASTER_USERNAME` y `MASTER_USER_PASSWORD` son las credenciales de inicio de sesión del usuario maestro de la instancia de base de datos en el clúster. En `CLUSTER_TYPE`, introduzca el tipo de clúster. Si especifica `single-node`, no necesitará el parámetro `NumberOfNodes`. El resto de los parámetros son opcionales.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Modificación de un clúster de Amazon Redshift

En este ejemplo se muestra cómo modificar la contraseña de usuario maestro de un clúster de Amazon Redshift usando AWS SDK for JavaScript. Para obtener más información sobre qué otra configuración puede modificar, consulte [ModifyCluster](#).

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `redshiftClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon Redshift. Sustituya **REGION** por su región de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `redshift-modify-cluster.js`. Asegúrese de configurar el SDK como se indicó anteriormente, incluida la instalación de los clientes y paquetes necesarios. Especifique la región de AWS, el nombre del clúster que desea modificar y la nueva contraseña de usuario maestro.

Note

Sustituya *CLUSTER_NAME* por el nombre del clúster y *MASTER_USER_PASSWORD* por la nueva contraseña de usuario maestro.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Visualización de los detalles de un clúster de Amazon Redshift

En este ejemplo, se muestra cómo consultar los detalles de un clúster de Amazon Redshift con AWS SDK for JavaScript. Para obtener más información sobre lo opcional, consulte [DescribeClusters](#).

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `redshiftClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon Redshift. Sustituya *REGION* por su región de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `redshift-describe-clusters.js`. Asegúrese de configurar el SDK como se indicó anteriormente, incluida la instalación de los clientes y paquetes necesarios. Especifique la región de AWS, el nombre del clúster que desea modificar y la nueva contraseña de usuario maestro.

Note

Sustituya `CLUSTER_NAME` por el nombre de su clúster.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminar un clúster de Amazon Redshift

En este ejemplo, se muestra cómo consultar los detalles de un clúster de Amazon Redshift con AWS SDK for JavaScript. Para obtener más información sobre qué otra configuración puede modificar, consulte [DeleteCluster](#).

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `redshiftClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon Redshift. Sustituya **REGION** por su región de 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 };
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `redshift-delete-clusters.js`. Asegúrese de configurar el SDK como se indicó anteriormente, incluida la instalación de los clientes y paquetes necesarios. Especifique la región de AWS, el nombre del clúster que desea modificar y la nueva contraseña de usuario maestro. Especifique si desea guardar una instantánea final del clúster antes de eliminarlo y, de ser así, el ID de la instantánea.

Note

Sustituya **CLUSTER_NAME** por el nombre de su clúster. Para ***SkipFinalClusterSnapshot***, especifique si desea crear una instantánea final del clúster antes de eliminarlo. Si especifica "false", especifique el ID de la instantánea final del clúster en **CLUSTER_SNAPSHOT_ID**. Puede obtener este ID haciendo clic en el enlace de la columna Instantáneas del clúster en el panel de control Clústeres y desplazándose hacia abajo hasta el panel Instantáneas. Tenga en cuenta que la raíz `rs:` no forma parte del ID de la instantánea.

```
// 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();
```

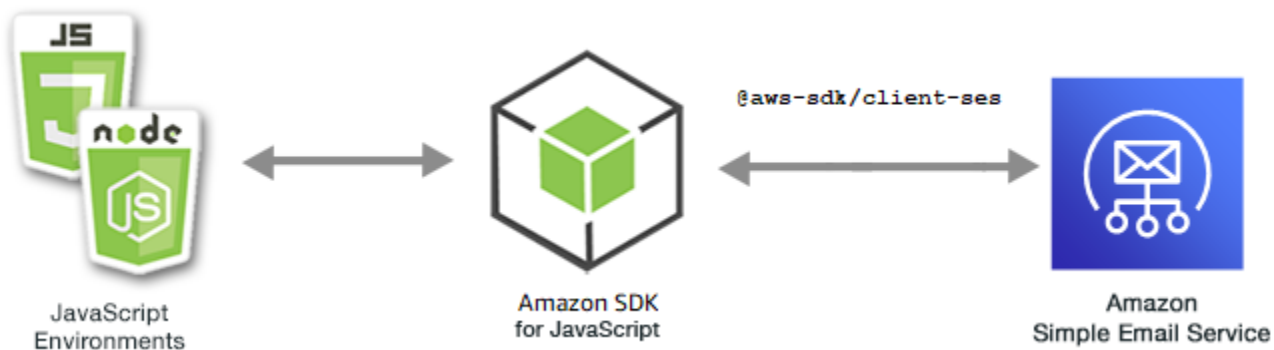
Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

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

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) es un servicio de envío de correos electrónicos con base en la nube y diseñado para ayudar a los responsables de marketing digital y a los desarrolladores de aplicaciones a enviar correos electrónicos de marketing, notificaciones y transacciones. Se trata de un servicio de confianza y rentable para negocios de todos los tamaños que usan el correo electrónico para mantenerse en contacto con sus clientes.



La JavaScript API de Amazon SES se expone a través de la clase de SES cliente. Para obtener más información acerca de cómo usar la clase de cliente de Amazon SES, consulte [Clase: SES](#) en la referencia de la API.

Temas

- [Administración de identidades en Amazon SES](#)
- [Trabajar con plantillas de correo electrónico en Amazon SES](#)
- [Envío de correo electrónico con Amazon SES](#)

Administración de identidades en Amazon SES



Este ejemplo de código de Node.js muestra:

- Cómo verificar direcciones de correo electrónico y dominios que se usan con Amazon SES.
- Cómo asignar una política AWS Identity and Access Management (IAM) a tus identidades de Amazon SES.
- Cómo enumerar todas las identidades de Amazon SES de tu AWS cuenta.
- Cómo eliminar identidades que se usan con Amazon SES.

En Amazon SES, una identidad es una dirección de correo electrónico o dominio que se utiliza para enviar correos electrónicos. Amazon SES requiere que verifique su dirección de correo electrónico o dominio para confirmar que es de su propiedad e impedir que otras personas los utilicen.

Para obtener información detallada acerca de cómo verificar direcciones de correo electrónico y dominios en Amazon SES, consulte [Verificación de direcciones de correo electrónico y dominios en Amazon SES](#) en la Guía para desarrolladores de Amazon Simple Email Service. Para obtener información acerca de cómo enviar autorizaciones en Amazon SES, consulte [Información general sobre el envío de autorizaciones en Amazon SES](#).

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para verificar y administrar identidades de Amazon SES. Los módulos de Node.js utilizan el SDK JavaScript para verificar las direcciones de correo electrónico y los dominios mediante los siguientes métodos de la clase SES cliente:

- [ListIdentitiesCommand](#)
- [DeleteIdentityCommand](#)
- [VerifyEmailIdentityCommand](#)
- [VerifyDomainIdentityCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos TypeScript ejemplos de nodos e instale los módulos necesarios AWS SDK for JavaScript y de terceros. Siga las instrucciones que figuran en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Mostrar sus identidades

En este ejemplo, utilice un módulo de Node.js para obtener una lista de direcciones de correo electrónico y dominios que se usarán con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_listidentities.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir el parámetro `IdentityType` y otros parámetros para el método `ListIdentitiesCommand` de la clase de cliente de Amazon SES. Para llamar al método `ListIdentitiesCommand`, invoque un objeto de servicio de Amazon SES transfiriendo el objeto de los parámetros.

Los data devueltos contienen una matriz de identidades de dominio tal y como especifica el parámetro `IdentityType`.

Note

Sustituya **IDENTITY_TYPE** por el tipo de identidad, que puede ser «» o `EmailAddress` «Domain».

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ses_listidentities.js
```

[Este código de ejemplo se puede encontrar aquí en GitHub](#)

Verificación de una identidad de dirección de correo electrónico

En este ejemplo, utilice un módulo de Node.js para comprobar los remitentes de direcciones de correo electrónico que se usarán con Amazon SES.


Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_verifyemailidentity.js`. Configure el SDK como se mostró anteriormente, incluida la descarga de los clientes y paquetes necesarios.

Cree un objeto para transferir el parámetro `EmailAddress` para el método `VerifyEmailIdentityCommand` de la clase de cliente de Amazon SES. Para llamar al método `VerifyEmailIdentityCommand`, invoque un objeto de servicio de cliente de Amazon SES transfiriendo los parámetros.

 Note

Sustituya *ADDRESS@DOMAIN.EXT* por la dirección de correo electrónico, como `name@example.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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El dominio se añade a Amazon SES para que se compruebe.

```
node ses_verifyemailidentity.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Verificación de una identidad de dominio

En este ejemplo, utilice un módulo de Node.js para comprobar los dominios de correo electrónico que se usarán con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_verifydomainidentity.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir el parámetro `Domain` para el método `VerifyDomainIdentityCommand` de la clase de cliente de Amazon SES. Para llamar al método `VerifyDomainIdentityCommand`, invoque un objeto de servicio de Amazon SES transfiriendo el objeto de los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya *AMI_ID* por el ID de la imagen de máquina de Amazon (AMI) que se va a ejecutar y *KEY_PAIR_NAME* del par de claves que se va a asignar al ID de la 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El dominio se añade a Amazon SES para que se compruebe.

```
node ses_verifydomainidentity.js
```

[Este código de ejemplo se puede encontrar aquí en GitHub](#)

Eliminación de identidades

En este ejemplo, utilice un módulo de Node.js para eliminar direcciones de correo electrónico o dominios que se usan con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_deleteidentity.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir el parámetro `Identity` para el método `DeleteIdentityCommand` de la clase de cliente de Amazon SES. Para llamar al método `DeleteIdentityCommand`, cree una `request` para invocar un objeto de servicio de cliente de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya **IDENTITY_TYPE** por el tipo de identidad que se va a eliminar y **IDENTITY_NAME** por el nombre de la identidad que se va a eliminar.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ses_deleteidentity.js
```

[Este código de ejemplo se puede encontrar aquí en GitHub](#)

Trabajar con plantillas de correo electrónico en Amazon SES



Este ejemplo de código de Node.js muestra:

- Cómo obtener una lista de todas sus plantillas de correo electrónico.
- Cómo recuperar y actualizar plantillas de correo electrónico.
- Cómo crear y eliminar plantillas de correo electrónico.

Con Amazon SES, puede enviar mensajes de correo electrónico personalizados mediante plantillas de correo electrónico. Para obtener más información sobre cómo crear y usar plantillas de correo electrónico en Amazon SES, consulte [Envío de correos electrónicos personalizados utilizando la API de Amazon SES](#) en la Guía para desarrolladores de Amazon Simple Email Service.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para trabajar con plantillas de correo electrónico. Los módulos de Node.js utilizan el SDK JavaScript para crear y utilizar plantillas de correo electrónico mediante los siguientes métodos de la clase SES cliente:

- [ListTemplatesCommand](#)
- [CreateTemplateCommand](#)
- [GetTemplateCommand](#)
- [DeleteTemplateCommand](#)
- [UpdateTemplateCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos TypeScript ejemplos de Node e instale los módulos necesarios AWS SDK for JavaScript y de terceros. Siga las instrucciones que figuran en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Obtención de una lista de plantillas de correo electrónico

En este ejemplo, utilice un módulo de Node.js para crear una plantilla de correo electrónico que se usará con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_listtemplates.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir los parámetros para el método `ListTemplatesCommand` de la clase de cliente de Amazon SES. Para llamar al método `ListTemplatesCommand`, invoque un objeto de servicio de cliente de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya *ITEMS_COUNT* por el número máximo de plantillas a obtener. El valor debe ser un mínimo de 1 y un valor máximo de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve la lista de plantillas.

```
node ses_listtemplates.js
```

[Este código de ejemplo se puede encontrar aquí en GitHub](#)

Obtención de una plantilla de correo electrónico

En este ejemplo, utilice un módulo de Node.js para obtener una plantilla de correo electrónico que se usará con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya *REGION* por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_gettemplate.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir el parámetro `TemplateName` para el método `GetTemplateCommand` de la clase de cliente de Amazon SES. Para llamar al método `GetTemplateCommand`, invoque un objeto de servicio de cliente de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya `TEMPLATE_NAME` por el nombre de la plantilla a obtener.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve los detalles de la plantilla.

```
node ses_gettemplate.js
```

[Este código de ejemplo se puede encontrar aquí en GitHub](#)

Creación de una plantilla de correo electrónico

En este ejemplo, utilice un módulo de Node.js para crear una plantilla de correo electrónico que se usará con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_createtemplate.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir los parámetros para el método `CreateTemplateCommand` de la clase de cliente de Amazon SES, incluidos `TemplateName`, `HtmlPart`, `SubjectPart` y `TextPart`. Para

llamar al método `CreateTemplateCommand`, invoque un objeto de servicio de cliente de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

En este ejemplo, se importan y utilizan los paquetes de clientes y comandos AWS de la V3 necesarios, y se utiliza el método siguiendo un patrón asíncrono/de espera. `send` En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya `TEMPLATE_NAME` por un nombre para la nueva plantilla, `HTML_CONTENT` por el contenido del correo electrónico etiquetado en HTML, `SUBJECT` por el asunto del correo electrónico y `TEXT_CONTENT` por el texto del correo electrónico.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. La plantilla se añade a Amazon SES.

```
node ses_createtemplate.js
```

[Este código de ejemplo se puede encontrar aquí en. GitHub](#)

Actualización de una plantilla de correo electrónico

En este ejemplo, utilice un módulo de Node.js para crear una plantilla de correo electrónico que se usará con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_updatetemplate.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir los valores de parámetro `Template` que desea actualizar en la plantilla, con el parámetro `TemplateName` obligatorio transferido al método `UpdateTemplateCommand` de la clase de cliente de Amazon SES. Para llamar al método `UpdateTemplateCommand`, invoque un objeto de servicio de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya `TEMPLATE_NAME` por un nombre para la nueva plantilla, `HTML_CONTENT` por el contenido del correo electrónico etiquetado en HTML, `SUBJECT` por el asunto del correo electrónico y `TEXT_CONTENT` por el texto del correo electrónico.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve los detalles de la plantilla.

```
node ses_updatetemplate.js
```

[Este código de ejemplo se puede encontrar aquí en. GitHub](#)

Eliminación de una plantilla de correo electrónico

En este ejemplo, utilice un módulo de Node.js para crear una plantilla de correo electrónico que se usará con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_delete_template.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir el parámetro `TemplateName` obligatorio al método `DeleteTemplateCommand` de la clase de cliente de Amazon SES. Para llamar al método `DeleteTemplateCommand`, invoque un objeto de servicio de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya `TEMPLATE_NAME` por el nombre de la plantilla que se va a eliminar.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve los detalles de la plantilla.

```
node ses_deletetemplate.js
```

[Este código de ejemplo se puede encontrar aquí en GitHub](#)

Envío de correo electrónico con Amazon SES



Este ejemplo de código de Node.js muestra:

- Enviar un correo electrónico en HTML o de texto.
- Enviar mensajes de correo electrónico basados en una plantilla de correo electrónico.
- Enviar mensajes de correo electrónico masivos basados en una plantilla de correo electrónico.

La API de Amazon SES ofrece dos formas distintas de enviar los correos electrónicos, según el nivel de control que desee tener sobre la composición del mensaje de correo electrónico: con o sin formato. Para obtener información detallada, consulte [Envío de correo electrónico con formato utilizando la API de Amazon SES](#) y [Envío de correo electrónico sin formato usando la API de Amazon SES](#).

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para enviar correos electrónicos de distintas maneras. Los módulos de Node.js utilizan el SDK JavaScript para crear y utilizar plantillas de correo electrónico mediante los siguientes métodos de la clase SES cliente:

- [SendEmailCommand](#)
- [SendTemplatedEmailCommand](#)
- [SendBulkTemplatedEmailCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos TypeScript ejemplos de Node e instale los módulos necesarios AWS SDK for JavaScript y de terceros. Siga las instrucciones que figuran en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).
- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Requisitos para enviar un mensaje de correo electrónico

Amazon SES crea un mensaje de correo electrónico e inmediatamente lo pone en la cola para su envío. Para enviar correo electrónico por medio del método `SendEmailCommand`, el mensaje debe cumplir los requisitos siguientes:

- Tiene que enviar el mensaje desde un dominio o una dirección de correo electrónico que se haya verificado. Si intenta enviar un correo electrónico mediante una dirección o un dominio que no se haya verificado, la operación generará un error "Email address not verified".

- Si su cuenta está todavía en el entorno de pruebas de Amazon SES, solo podrá realizar envíos a direcciones o dominios verificados, o a direcciones de correo electrónico que estén asociadas al simulador de bandeja de correo de Amazon SES. Para obtener más información, consulte [Verificación de direcciones de correo electrónico y dominios](#) en la Guía para desarrolladores de Amazon Simple Email Service.
- El tamaño total del mensaje, los archivos adjuntos incluidos, debe ser inferior a 10 MB.
- El mensaje tiene que incluir al menos una dirección de correo electrónico de destinatario. La dirección del destinatario puede ser una dirección A:, una dirección CC: o una dirección CCO. Si la dirección de correo electrónico de un destinatario no es válida (es decir, no sigue el formato `UserName@[SubDomain.]Domain.TopLevelDomain`), se rechazará todo el mensaje, aunque este contenga otros destinatarios que sí son válidos.
- El mensaje no puede incluir más de 50 destinatarios entre los campos A:, CC: y CCO: en total. Si necesita enviar un mensaje de correo electrónico a más destinatarios, puede dividir la lista de destinatarios en grupos de 50 o menos y luego llamar al método `sendEmail` varias veces para enviar el mensaje a cada grupo.

Envío de un correo electrónico

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_sendemail.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir los valores de parámetros que definen el correo electrónico que se va a enviar, como las direcciones del remitente y el receptor, y el cuerpo del correo electrónico

en formatos de texto y HTML, al método `SendEmailCommand` de la clase de cliente de Amazon SES. Para llamar al método `SendEmailCommand`, invoque un objeto de servicio de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya `RECEIVER_ADDRESS` por la dirección a la que se va a enviar el correo electrónico y `SENDER_ADDRESS` por la dirección de correo electrónico desde la que se envía el correo electrónico.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El correo electrónico se pone en cola para que Amazon SES lo envíe.

```
node ses_sendemail.js
```

[Este código de ejemplo se puede encontrar aquí en. GitHub](#)

Envío de un correo electrónico mediante una plantilla

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_sendtemplatedemail.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir los valores de parámetros que definen el correo electrónico que se va a enviar, como las direcciones del remitente y el receptor, el cuerpo del correo electrónico en formatos de texto y HTML, al método `SendTemplatedEmailCommand` de la clase de cliente de Amazon SES. Para llamar al método `SendTemplatedEmailCommand`, invoque un objeto de servicio de cliente de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya `REGION` por su AWS región, `RECEIVER_ADDRESS` por la dirección a la que enviar el correo electrónico, `SENDER_ADDRESS` por la dirección de correo electrónico desde la que se envía el correo electrónico y `TEMPLATE_NAME` por el nombre de la plantilla.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El correo electrónico se pone en cola para que Amazon SES lo envíe.

```
node ses_sendtemplatedemail.js
```

[Puedes encontrar este código de ejemplo aquí en GitHub](#)

Envío masivo de correos electrónicos mediante una plantilla

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES.

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `sesClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `ses_sendbulktemplatedemail.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir los valores de parámetros que definen el correo electrónico que se va a enviar, como las direcciones del remitente y el receptor, y el cuerpo del correo electrónico en formatos de texto y HTML, al método `SendBulkTemplatedEmailCommand` de la clase de cliente de Amazon SES. Para llamar al método `SendBulkTemplatedEmailCommand`, invoque un objeto de servicio de Amazon SES transfiriendo los parámetros.

Note

En este ejemplo, se importan y se utilizan los paquetes de clientes y comandos de la versión 3 necesarios del AWS servicio, y se utiliza el `send` método siguiendo un patrón asíncrono o de espera. En su lugar, puede crear este ejemplo con los comandos de la V2 realizando algunos cambios menores. Para obtener más detalles, consulte [Uso de los comandos de la versión 3](#).

Note

Sustituya `RECEIVER_ADDRESS` por la dirección a la que se va a enviar el correo electrónico y `SENDER_ADDRESS` por la dirección de correo electrónico desde la que se envía el correo electrónico.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El correo electrónico se pone en cola para que Amazon SES lo envíe.

```
node ses_sendbulktemplatedemail.js
```

[Este código de ejemplo se puede encontrar aquí en GitHub](#)

Ejemplos de Amazon Simple Notification Service

Amazon Simple Notification Service (Amazon SNS) es un servicio web que coordina y gestiona la entrega o el envío de mensajes a los puntos de enlace o clientes suscritos.

En Amazon SNS, existen dos tipos de clientes, publicadores y suscriptores, también conocidos como productores y consumidores.



Los publicadores se comunican de forma asíncrona con los suscriptores generando y enviando un mensaje a un tema, que es un punto de acceso lógico y un canal de comunicación. Los suscriptores (servidores web, direcciones de correo electrónico, colas de Amazon SQS, funciones de Lambda) consumen o reciben el mensaje o la notificación por medio de uno de los protocolos admitidos (Amazon SQS, HTTP/S, email, SMS, Lambda) cuando están suscritos al tema.

La API de JavaScript para Amazon SNS se expone a través de la [Clase: SNS](#).

Temas

- [Administración de temas en Amazon SNS](#)
- [Publicación de mensajes en Amazon SNS](#)
- [Administración de suscripciones en Amazon SNS](#)
- [Envío de mensajes SMS con Amazon SNS](#)

Administración de temas en Amazon SNS



Este ejemplo de código de Node.js muestra:

- Cómo crear temas en Amazon SNS en los que pueda publicar notificaciones.
- Cómo eliminar temas creados en Amazon SNS.
- Cómo obtener una lista de los temas disponibles.
- Cómo obtener y establecer atributos de temas.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para crear, enumerar y eliminar temas de Amazon SNS y para gestionar atributos de los temas. Los módulos de Node.js usan el SDK para JavaScript para administrar temas mediante los métodos de clase de cliente de SNS siguientes:

- [CreateTopicCommand](#)
- [ListTopicsCommand](#)
- [DeleteTopicCommand](#)
- [GetTopicAttributesCommand](#)
- [SetTopicAttributesCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

⚠ Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js.](#)
- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript.](#)

Creación de un tema

En este ejemplo, utilice un módulo de Node.js para crear un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `create-topic.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto para transferir el Name del nuevo tema al método `CreateTopicCommand` de la clase de cliente de Amazon SNS. Para llamar al método `CreateTopicCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros. Los data devueltos contienen el ARN del tema.

📘 Note

Reemplace **TOPIC_NAME** por el nombre del tema.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node create-topic.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Enumeración de los temas de

En este ejemplo, utilice un módulo de Node.js para generar una lista de todos los temas de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `list-topics.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto vacío para transferirlo al método `ListTopicsCommand` de la clase de cliente de Amazon SNS. Para llamar al método `ListTopicsCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros. Los data devueltos contienen una matriz de Nombres de recursos de Amazon (ARN) de su tema.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node list-topics.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Eliminación de un tema

En este ejemplo, utilice un módulo de Node.js para eliminar un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `delete-topic.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto que contenga el `TopicArn` del tema que se va a eliminar para transferirlo al método `DeleteTopicCommand` de la clase de cliente Amazon SNS. Para llamar al método `DeleteTopicCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

Reemplace **TOPIC_ARN** por el Nombre de recurso de Amazon (ARN) del tema que va a eliminar.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node delete-topic.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Obtención de atributos de temas

En este ejemplo, utilice un módulo de Node.js para recuperar atributos de un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `get-topic-attributes.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el `TopicArn` de un tema que se vaya a eliminar para transferirlo al método `GetTopicAttributesCommand` de la clase de cliente de Amazon SNS. Para llamar al

método `GetTopicAttributesCommand`, invoque un objeto de servicio de cliente de Amazon SNS, mediante el traspaso del objeto de parámetros.

Note

Sustituya `TOPIC_ARN` por el ARN del tema.

```
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: 'xxxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx: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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node get-topic-attributes.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Configuración de los atributos de un tema

En este ejemplo, utilice un módulo de Node.js para establecer los atributos mutables de un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `set-topic-attributes.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga los parámetros para realizar la actualización del atributo, como el `TopicArn` del tema cuyos atributos desea establecer, el nombre del atributo que se va a establecer y el nuevo valor para dicho atributo. Solo puede establecer los atributos `Policy`, `DisplayName` y `DeliveryPolicy`. Transfiera los parámetros al método `SetTopicAttributesCommand` de la clase de cliente de Amazon SNS. Para llamar al método `SetTopicAttributesCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

Sustituya `ATTRIBUTE_NAME` por el nombre del atributo que está configurando, `TOPIC_ARN` por el Nombre de recurso de Amazon (ARN) del tema cuyos atributos desee establecer y `NEW_ATTRIBUTE_VALUE` por el nuevo valor de ese 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node set-topic-attributes.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Publicación de mensajes en Amazon SNS



Este ejemplo de código de Node.js muestra:

- Cómo publicar mensajes en un tema de Amazon SNS.

El escenario

En este ejemplo va a utilizar una serie de módulos de Node.js para publicar mensajes de Amazon SNS en puntos de conexión de temas, correos electrónicos o números de teléfono. Los módulos de Node.js usan el SDK para JavaScript para enviar mensajes mediante este método de la clase de cliente de SNS:

- [PublishCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js.](#)
- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript.](#)

Publicación de un mensaje en un tema de SNS

En este ejemplo, utilice un módulo de Node.js para publicar un mensaje en un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `publish-topic.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga los parámetros para la publicación de un mensaje, incluido el texto del mensaje y el nombre de recurso de Amazon (ARN) del tema de Amazon SNS. Para obtener información detallada sobre los atributos de SMS, consulte [SetSMSAttributes](#).

Pase los parámetros al método `PublishCommand` de la clase de cliente de Amazon SNS. Cree una función asíncrona que invoque un objeto de servicio de cliente de Amazon SNS, mediante el traspaso del objeto de parámetros.

Note

Sustituya **MESSAGE_TEXT** por el texto del mensaje y **TOPIC_ARN** por el ARN del tema de 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-xxxxxxxxxxxxxx'
  // }
  return response;
};
```

Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node publish-topic.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Administración de suscripciones en Amazon SNS



Este ejemplo de código de Node.js muestra:

- Cómo mostrar una lista de todas las suscripciones en un tema de Amazon SNS.
- Cómo suscribir una dirección de correo electrónico, un punto de conexión de una aplicación o una función de Lambda a un tema de Amazon SNS.
- Cómo cancelar una suscripción a temas de Amazon SNS.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para publicar mensajes de notificación en temas de Amazon SNS. Los módulos de Node.js usan el SDK para JavaScript para administrar temas mediante los métodos de clase de cliente de Amazon SNS siguientes:

- [ListSubscriptionsByTopicCommand](#)
- [SubscribeCommand](#)
- [ConfirmSubscriptionCommand](#)
- [UnsubscribeCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

⚠ Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js.](#)
- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript.](#)

Generación de una lista de suscripciones a un tema

En este ejemplo, utilice un módulo de Node.js para generar una lista de todas las suscripciones a un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `list-subscriptions-by-topic.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el parámetro `TopicArn` para el tema cuyas suscripciones desee incluir en la lista. Transfiera los parámetros al método `ListSubscriptionsByTopicCommand` de la clase de cliente de Amazon SNS. Para llamar al método `ListSubscriptionsByTopicCommand`, cree una función asíncrona que invoque un objeto de servicio de cliente de Amazon SNS y transfiera el objeto de los parámetros.

Note

Reemplace *TOPIC_ARN* por el nombre de recurso de Amazon (ARN) del tema cuyas suscripciones quiera incluir.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node list-subscriptions-by-topic.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Suscripción de una dirección de correo electrónico a un tema

En este ejemplo, va a utilizar un módulo de Node.js para suscribir una dirección de correo electrónico para que reciba mensajes de correo electrónico SMTP de un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `subscribe-email.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el parámetro `Protocol` para especificar el protocolo de email, el `TopicArn` del tema al que se suscribirá y una dirección de correo electrónico como el mensaje `Endpoint`. Transfiera los parámetros al método `SubscribeCommand` de la clase de cliente de Amazon SNS. Puede utilizar el método `subscribe` para suscribir varios puntos de conexión diferentes a un tema de Amazon SNS, en función de los valores que se utilicen para los parámetros que se transfieran, tal y como se verá en otros ejemplos de este tema.

Para llamar al método `SubscribeCommand`, cree una función asíncrona que invoque un objeto de servicio de cliente de Amazon SNS y transfiera el objeto de los parámetros.

Note

Sustituya **TOPIC_ARN** por el Nombre de recurso de Amazon (ARN) para el tema y **EMAIL_ADDRESS** por la dirección de correo electrónico a la que desea suscribirse.

```
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 = "user@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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node subscribe-email.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Confirmar suscripciones

En este ejemplo, use un módulo de Node.js para verificar la intención del propietario de un punto de conexión de recibir mensajes validando el token enviado al punto de conexión por una acción de suscripción previa.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él para crear el objeto de cliente de Amazon SES. Sustituya **REGION** por su región de AWS.

```
import { SNSClient } from "@aws-sdk/client-sns";

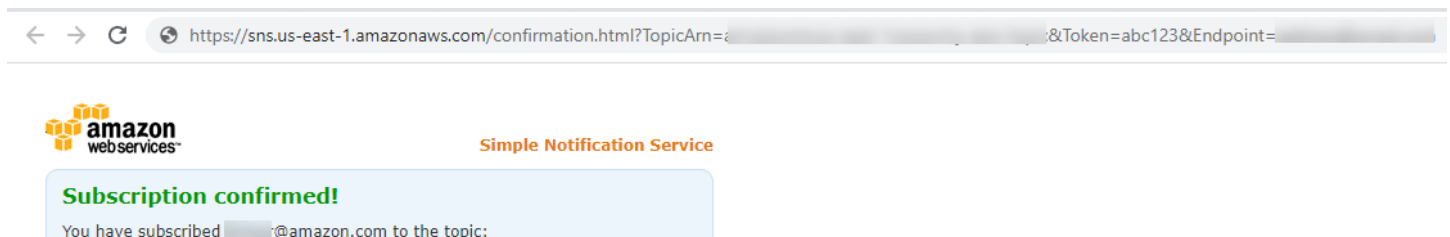
// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `confirm-subscription.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Defina los parámetros, incluidos `TOPIC_ARN` y `TOKEN`, y defina un valor de `TRUE` o `FALSE` para `AuthenticateOnUnsubscribe`.

El token es un token de corta duración que se envía al propietario de un punto de conexión durante una acción `SUBSCRIBE` anterior. Por ejemplo, en el caso de un punto de conexión de correo electrónico, el `TOKEN` está en la URL del correo electrónico de confirmación de suscripción enviado al propietario del correo electrónico. Por ejemplo, `abc123` es el token en la siguiente URL:



Para llamar al método `ConfirmSubscriptionCommand`, cree una función asíncrona para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

Sustituya *TOPIC_ARN* por el Nombre de recurso de Amazon (ARN) del tema, *TOKEN* por el valor del token de la URL enviada al propietario del punto de conexión en una acción de *Subscribe* anterior y defina *AuthenticateOnUnsubscribe* con un valor de TRUE o 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:xxxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxxx'
// }
return response;
};
```

Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node confirm-subscription.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Suscripción de un punto de enlace de aplicación a un tema

En este ejemplo, va a utilizar un módulo de Node.js para suscribir un punto de enlace de una aplicación móvil para que reciba notificaciones de un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `subscribe-app.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los módulos y paquetes necesarios.

Cree un objeto que contenga el parámetro `Protocol` para especificar el protocolo de la `application`, el `TopicArn` del tema al que va a suscribirse y el Nombre de recurso de Amazon (ARN) del punto de conexión de la aplicación móvil para el parámetro `Endpoint`. Transfiera los parámetros al método `SubscribeCommand` de la clase de cliente de Amazon SNS.

Para llamar al método `SubscribeCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

Sustituya *TOPIC_ARN* por el nombre de recurso de Amazon (ARN) del tema y *MOBILE_ENDPOINT_ARN* por el punto de conexión al que está suscribiendo al tema.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node subscribe-app.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Suscripción de una función de Lambda a un tema

En este ejemplo, va a utilizar un módulo de Node.js para suscribir una función para que reciba notificaciones de un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `subscribe-lambda.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el parámetro `Protocol` y que especifique el protocolo `lambda`, el `TopicArn` del tema al que va a suscribirse y el Nombre de recurso de Amazon (RN) de una función AWS Lambda como el parámetro `Endpoint`. Transfiera los parámetros al método `SubscribeCommand` de la clase de cliente de Amazon SNS.

Para llamar al método `SubscribeCommand`, cree una función asíncrona para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

Sustituya **TOPIC_ARN** por el nombre de recurso de Amazon (ARN) del tema y **LAMBDA_FUNCTION_ARN** por el nombre de recurso de Amazon (ARN) de la función de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node subscribe-lambda.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cancelación de la suscripción a un tema

En este ejemplo, utilice un módulo de Node.js para cancelar la suscripción a un tema de Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `unsubscribe.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios.

Cree un objeto que contenga el parámetro `SubscriptionArn` que especifique el Nombre de recurso de Amazon (ARN) de la suscripción que desea cancelar. Transfiera los parámetros al método `UnsubscribeCommand` de la clase de cliente de Amazon SNS.

Para llamar al método `UnsubscribeCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

Reemplace **TOPIC_SUBSCRIPTION_ARN** por el nombre de recurso de Amazon (ARN) de la suscripción para cancelar la suscripción.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node unsubscribe.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Envío de mensajes SMS con Amazon SNS



Este ejemplo de código de Node.js muestra:

- Cómo obtener y establecer las preferencias de mensajería de SMS para Amazon SNS.
- Cómo comprobar un número de teléfono para ver si se ha desactivado la opción de recepción de mensajes SMS.
- Cómo obtener una lista de números de teléfono con la opción de recepción de mensajes SMS desactivada
- Cómo enviar un mensaje SMS.

El escenario

Puede utilizar Amazon SNS para enviar mensajes de texto o mensajes SMS a dispositivos habilitados para recibir SMS. Dispone de la capacidad de enviar un mensaje directamente a un número de teléfono o de enviar un mensaje a varios números de teléfono a la vez suscribiendo dichos números de teléfono a un tema y enviando el mensaje al tema.

En este ejemplo, va a utilizar una serie de módulos de Node.js para publicar mensajes de texto SMS de Amazon SNS a dispositivos habilitados para SMS. Los módulos de Node.js usan el SDK para JavaScript para publicar mensajes SMS mediante los métodos de clase de cliente de SNS siguientes:

- [GetSMSAttributesCommand](#)
- [SetSMSAttributesCommand](#)
- [CheckIfPhoneNumberIsOptedOutCommand](#)
- [ListPhoneNumbersOptedOutCommand](#)
- [PublishCommand](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).
- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Obtención de atributos de SMS

Utilice Amazon SNS para especificar las preferencias de mensajería SMS, como la forma en que se optimizan sus envíos (por coste o por fiabilidad de la entrega), su límite de gasto mensual, cómo se registran los envíos de mensajes y si desea suscribirse a informes de uso de SMS diarios. Estas preferencias se recuperan y se establecen como atributos SMS para Amazon SNS.

En este ejemplo, va a utilizar un módulo de Node.js para obtener los atributos de SMS actuales en Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `get-sms-attributes.js`.

Configure el SDK como se mostró anteriormente, incluida la descarga de los clientes y paquetes necesarios. Cree un objeto que contenga los parámetros para obtener atributos de SMS, como los nombres de los atributos individuales que desea obtener. Para obtener más información sobre los atributos de SMS disponibles, consulte [SetSMSAttributes](#) en la Referencia de la API de Amazon Simple Notification Service.

Este ejemplo obtiene el atributo `DefaultSMSType` que controla si se envían mensajes SMS como `Promotional`, que optimiza la entrega de mensajes para conseguir el costo más bajo, o como `Transactional`, que optimiza el envío de mensajes para conseguir la máxima fiabilidad. Transfiera los parámetros al método `SetTopicAttributesCommand` de la clase de cliente de Amazon SNS. Para llamar al método `SetSMSAttributesCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

Sustituya *ATTRIBUTE_NAME* por el nombre del 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node get-sms-attributes.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Configuración de atributos de SMS

En este ejemplo, va a utilizar un módulo de Node.js para obtener los atributos de SMS actuales en Amazon SNS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `set-sms-attribute-type.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios. Cree un objeto que contenga los parámetros para establecer atributos de SMS, como los nombres de los atributos individuales que desea establecer y los valores que se establecerán para cada uno de ellos. Para todos los atributos de SMS, consulte [SetSMSAttributes](#) en la Referencia de la API de Amazon Simple Notification Service.

Este ejemplo establece el atributo `DefaultSMSType` en `Transactional`, que optimiza el envío de mensajes para conseguir la máxima fiabilidad. Transfiera los parámetros al método `SetTopicAttributesCommand` de la clase de cliente de Amazon SNS. Para llamar al método `SetSMSAttributesCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node set-sms-attribute-type.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Comprobación de si se ha desactivado un número de teléfono

En este ejemplo, va a utilizar un módulo de Node.js para comprobar si un número de teléfono tiene desactivada la opción de recibir mensajes SMS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `check-if-phone-number-is-opted-out.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto que contenga el número de teléfono que se desea comprobar como parámetro.

Este ejemplo establece el parámetro `PhoneNumber` para especificar el número de teléfono que se va a comprobar. Transfiera el objeto al método `CheckIfPhoneNumberIsOptedOutCommand` de la clase de cliente de Amazon SNS. Para llamar al método `CheckIfPhoneNumberIsOptedOutCommand`, cree una función asíncronica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

1.

Sustituya *PHONE_NUMBER* por el número de teléfono.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node check-if-phone-number-is-opted-out.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Generación de una lista de los números de teléfono desactivados

En este ejemplo, va a utilizar un módulo de Node.js para obtener una lista de números de teléfono que tienen desactivada la opción de recibir mensajes SMS.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `list-phone-numbers-opted-out.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto vacío como parámetro.

Transfiera el objeto al método `ListPhoneNumbersOptedOutCommand` de la clase de cliente de Amazon SNS. Para llamar al método `ListPhoneNumbersOptedOutCommand`, cree una función asíncrona para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

```
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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node list-phone-numbers-opted-out.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Publicación de un mensaje SMS

En este ejemplo, utilice un módulo de Node.js para enviar un mensaje SMS a un número de teléfono.

Cree un directorio `libs` y cree un módulo Node.js con el nombre de archivo `snsClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente de Amazon SNS. Sustituya **REGION** por su región de 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({});
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `publish-sms.js`. Configure el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios. Cree un objeto que contenga los parámetros `Message` y `PhoneNumber`.

Cuando envíe un mensaje SMS, especifique el número de teléfono usando la formato E.164.

E.164 es un estándar de estructura de número de teléfono utilizado para las telecomunicaciones internacionales. Los números que aplican este formato pueden tener un máximo de 15 dígitos y van prefijados con el carácter (+) y el código de país. Por ejemplo, un número de teléfono de los EE. UU. en formato E.164 se mostraría como +1001XXX5550100.

En este ejemplo se establece el parámetro `PhoneNumber` para especificar el número de teléfono al que se enviará el mensaje. Transfiera el objeto al método `PublishCommand` de la clase de cliente de Amazon SNS. Para llamar al método `PublishCommand`, cree una función asincrónica para invocar un objeto de servicio de Amazon SNS mediante el traspaso del objeto de parámetros.

Note

Sustituya `TEXT_MESSAGE` por el mensaje de texto y `PHONE_NUMBER` por el número de teléfono.

```
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-xxxxxxxxxxxxxxx'  
// }  
return response;  
};
```

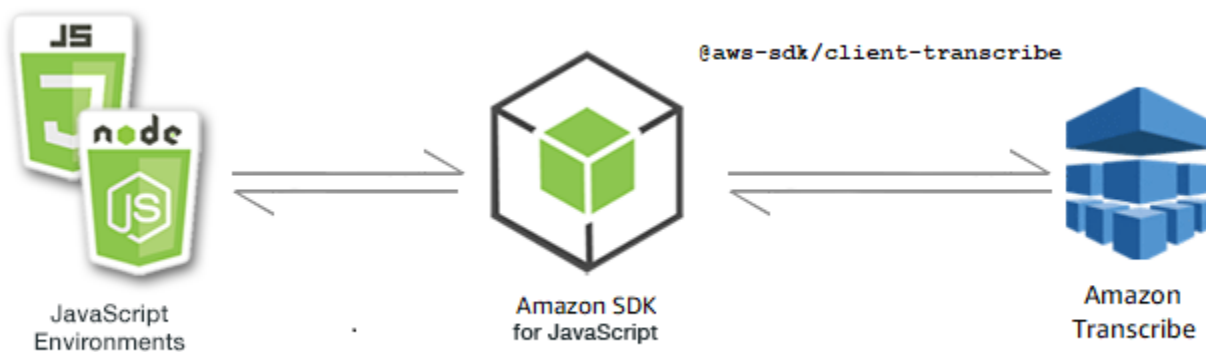
Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node publish-sms.js
```

Este código de muestra se puede encontrar [aquí en GitHub](#).

Ejemplos de Amazon Transcribe

Amazon Transcribe permite a los desarrolladores añadir fácilmente a sus aplicaciones capacidades de conversión de voz a texto en tiempo real.



La JavaScript API de Amazon Transcribe se expone a través de la clase de [TranscribeService](#) cliente.

Temas

- [Ejemplos de Amazon Transcribe](#)
- [Ejemplos de Amazon Transcribe Medical](#)

Ejemplos de Amazon Transcribe

En este ejemplo, se utilizan una serie de módulos de Node.js para crear, enumerar y eliminar trabajos de transcripción mediante los siguientes métodos de la clase de cliente de TranscribeService:

- [StartTranscriptionJobCommand](#)
- [ListTranscriptionJobsCommand](#)
- [DeleteTranscriptionJobCommand](#)

Para obtener más información acerca de los usuarios de Amazon Transcribe, consulte la [Guía para desarrolladores de Amazon Transcribe](#).

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos TypeScript ejemplos de nodos e instale los módulos necesarios AWS SDK for JavaScript y de terceros. Siga las instrucciones de [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).
- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#)

Cómo iniciar un trabajo de Amazon Transcribe

En este ejemplo se muestra cómo iniciar un trabajo de transcripción de Amazon Transcribe con AWS SDK for JavaScript. Para obtener más información, consulte [StartTranscriptionJobCommand](#).

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `transcribeClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente Amazon Transcribe. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `transcribe-create-job.js`. Asegúrese de configurar el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios. Cree un objeto de parámetros especificando los parámetros necesarios. Inicie el trabajo mediante el comando `StartMedicalTranscriptionJobCommand`.

Note

Sustituya **MEDICAL_JOB_NAME** por un nombre para el trabajo de transcripción. En **OUTPUT_BUCKET_NAME**, especifique el bucket de Amazon S3 en el que se guardará la salida. En **JOB_TYPE**, especifique los tipos de trabajo. En **SOURCE_LOCATION**, especifique la ubicación del archivo de origen. En **SOURCE_FILE_LOCATION**, especifique la ubicación del archivo multimedia 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node transcribe-create-job.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtener una lista de trabajos de Amazon Transcribe


Este ejemplo muestra cómo obtener una lista de los trabajos de transcripción de Amazon Transcribe utilizando AWS SDK for JavaScript. Para obtener más información sobre qué otra configuración puede modificar, consulte [ListTranscriptionJobCommand](#).

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `transcribeClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente Amazon Transcribe. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `transcribe-list-jobs.js`. Asegúrese de configurar el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios. Cree un objeto de parámetros con los parámetros requeridos.

 Note

Sustituya **KEY_WORD** por una palabra clave que tenga que contener el nombre de los trabajos devueltos.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node transcribe-list-jobs.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminar un trabajo de Amazon Transcribe

En este ejemplo, se muestra cómo eliminar un trabajo de transcripción de Amazon Transcribe utilizando AWS SDK for JavaScript. Para obtener más información sobre los elementos opcionales, consulte [DeleteTranscriptionJobCommand](#).

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `transcribeClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente Amazon Transcribe. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `transcribe-delete-job.js`. Asegúrese de configurar el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios. Especifique la AWS región y el nombre del trabajo que desea eliminar.

Note

Sustituya **JOB_NAME** por el nombre del trabajo que se va a eliminar.

```
// 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, 'transcription_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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node transcribe-delete-job.js
```

Puede encontrar este código de ejemplo [aquí en GitHub](#).

Ejemplos de Amazon Transcribe Medical

En este ejemplo, se utilizan una serie de módulos de Node.js para crear, enumerar y eliminar trabajos de transcripción médica mediante los siguientes métodos de la clase de cliente de TranscribeService:

- [StartMedicalTranscriptionJobCommand](#)
- [ListMedicalTranscriptionJobsCommand](#)
- [DeleteMedicalTranscriptionJobCommand](#)

Para obtener más información acerca de los usuarios de Amazon Transcribe, consulte la [Guía para desarrolladores de Amazon Transcribe](#).

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos TypeScript ejemplos de Node e instale los módulos necesarios AWS SDK for JavaScript y de terceros. Siga las instrucciones de [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de SDK y herramientas de AWS.

⚠ Important

Estos ejemplos muestran cómo importar/exportar comandos y objetos del servicio de cliente mediante ECMAScript6 (ES6).

- Requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).
- Si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#)

Cómo iniciar un trabajo de transcripción de Amazon Transcribe Medical

En este ejemplo se muestra cómo iniciar un trabajo de transcripción de Amazon Transcribe Medical utilizando AWS SDK for JavaScript. Para obtener más información, consulte [start MedicalTranscription Job](#).

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `transcribeClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente Amazon Transcribe. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `transcribe-create-medical-job.js`. Asegúrese de configurar el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios. Cree un objeto de parámetros especificando los parámetros necesarios. Inicie el trabajo médico con el comando `StartMedicalTranscriptionJobCommand`.

📘 Note

Sustituya **MEDICAL_JOB_NAME** por un nombre para el trabajo de transcripción médica. En **OUTPUT_BUCKET_NAME**, especifique el bucket de Amazon S3 en el que se guardará la salida. En **JOB_TYPE**, especifique los tipos de trabajo. En **SOURCE_LOCATION**, especifique la

ubicación del archivo de origen. En *SOURCE_FILE_LOCATION*, especifique la ubicación del archivo multimedia 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node transcribe-create-medical-job.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtener una lista de trabajos de Amazon Transcribe Medical

En este ejemplo, se muestra cómo obtener una lista de los trabajos de transcripción de Amazon Transcribe utilizando AWS SDK for JavaScript. Para obtener más información, consulte [ListTranscriptionMedicalJobsComando](#).

Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `transcribeClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente Amazon Transcribe. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `transcribe-list-medical-jobs.js`. Asegúrese de configurar el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes necesarios. Cree un objeto de parámetros con los parámetros necesarios y obtenga una lista de los trabajos médicos utilizando el comando `ListMedicalTranscriptionJobsCommand`.

Note

Sustituya **KEYWORD** por una palabra clave que tenga que contener el nombre del trabajo devuelto.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node transcribe-list-medical-jobs.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminar un trabajo de Amazon Transcribe Medical

En este ejemplo, se muestra cómo eliminar un trabajo de transcripción de Amazon Transcribe utilizando AWS SDK for JavaScript. Para obtener más información sobre los elementos opcionales, consulte [DeleteTranscriptionMedicalJobCommand](#).


Cree un directorio `libs` y un módulo Node.js con el nombre de archivo `transcribeClient.js`. Copie y pegue el siguiente código en él, para crear el objeto de cliente Amazon Transcribe. Sustituya **REGION** por su AWS región.

```
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 };
```

Puedes encontrar este código de ejemplo [aquí en GitHub](#).

Cree un módulo de Node.js con el nombre de archivo `transcribe-delete-job.js`. Asegúrese de configurar el SDK como se mostró anteriormente, incluida la instalación de los clientes y paquetes

necesarios. Cree un objeto de parámetros con los parámetros necesarios y elimine el trabajo médico utilizando el comando `DeleteMedicalJobCommand`.

 Note

Sustituya *JOB_NAME* por el nombre del trabajo que se va a eliminar.

```
// 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_transcription_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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node transcribe-delete-medical-job.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Configuración de Node.js en una instancia de Amazon EC2

Un escenario habitual para usar Node.js con el SDK JavaScript es configurar y ejecutar una aplicación web Node.js en una instancia de Amazon Elastic Compute Cloud (Amazon EC2). En este tutorial, creará una instancia Linux, se conectará a ella mediante SSH y, a continuación, instalará Node.js para ejecutarse en dicha instancia.

Requisitos previos

En este tutorial se presupone que ya ha lanzado una instancia Linux con un nombre de DNS público al que se puede tener acceso desde Internet y al que se puede conectar a través de SSH. Para obtener más información, consulte [Paso 1: Lanzamiento de una instancia](#) en la Guía del usuario de Amazon EC2.

Important

Utilice Imagen de máquina de Amazon (AMI) de Amazon Linux 2023 al ejecutar una nueva instancia de Amazon EC2.

También debe haber configurado el grupo de seguridad para que permita las conexiones SSH (puerto 22), HTTP (puerto 80) y HTTPS (puerto 443). Para obtener más información sobre estos requisitos previos, consulte [Configuración con Amazon](#) EC2 en la Guía del usuario de Amazon EC2.

Procedimiento

El siguiente procedimiento le será útil para instalar Node.js en una instancia Amazon Linux. Puede utilizar este servidor para alojar una aplicación web de Node.js.

Para configurar Node.js en su instancia Linux

1. Conecte su instancia Linux como `ec2-user` mediante SSH.
2. Instale el administrador de versiones de nodos (nvm) escribiendo lo siguiente en la línea de comandos.

⚠ Warning

AWS no controla el siguiente código. Antes de ejecutarlo, asegúrese de comprobar su autenticidad e integridad. Puede encontrar más información sobre este código en el GitHub repositorio [nvm](#).

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

Utilizaremos nvm para instalar Node.js, ya que nvm puede instalar varias versiones de Node.js y permitirle alternar entre ellas.

3. Cargue nvm escribiendo lo siguiente en la línea de comandos.

```
source ~/.bashrc
```

4. Utilice nvm para instalar la última versión de LTS escribiendo lo siguiente en la línea de comandos.

```
nvm install --lts
```

Si instala Node.js también instalará el administrador de paquetes de nodos (npm) para poder instalar módulos adicionales según sea necesario.

5. Compruebe que Node.js esté instalado y ejecutándose correctamente, escribiendo lo siguiente en la línea de comandos.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Esto presenta el siguiente mensaje que muestra la versión de Node.js que se está ejecutando.

Running Node.js *VERSION*

📘 Note

La instalación del nodo solo se aplica a la sesión actual de Amazon EC2. Si reinicia la sesión de CLI, debe volver a usar nvm para habilitar la versión de nodo instalada. Si la instancia finaliza, debe volver a instalar el nodo. La alternativa es crear una imagen de máquina de

Amazon (AMI) de la instancia de Amazon EC2 una vez que tenga la configuración que desea conservar, como se describe en el siguiente tema.

Creación de una Imagen de máquina de Amazon (AMI)

Después de instalar Node.js en una instancia de Amazon EC2, puede crear una imagen de máquina de Amazon (AMI) a partir de dicha instancia. Si crea una AMI le será más fácil aprovisionar varias instancias de Amazon EC2 con la misma instalación de Node.js. Para obtener más información sobre la creación de una AMI a partir de una instancia existente, consulte [Creación de una AMI de Linux respaldada por Amazon EBS](#) en la Guía del usuario de Amazon EC2.

Recursos relacionados

Para obtener más información acerca de los comandos y el software que se utilizan en este tema, visite las siguientes páginas web:

- [Administrador de versiones de nodos \(nvm\): consulte nvm repo en. GitHub](#)
- Administrador de paquetes de nodos (npm): consulte el [sitio web de npm](#).

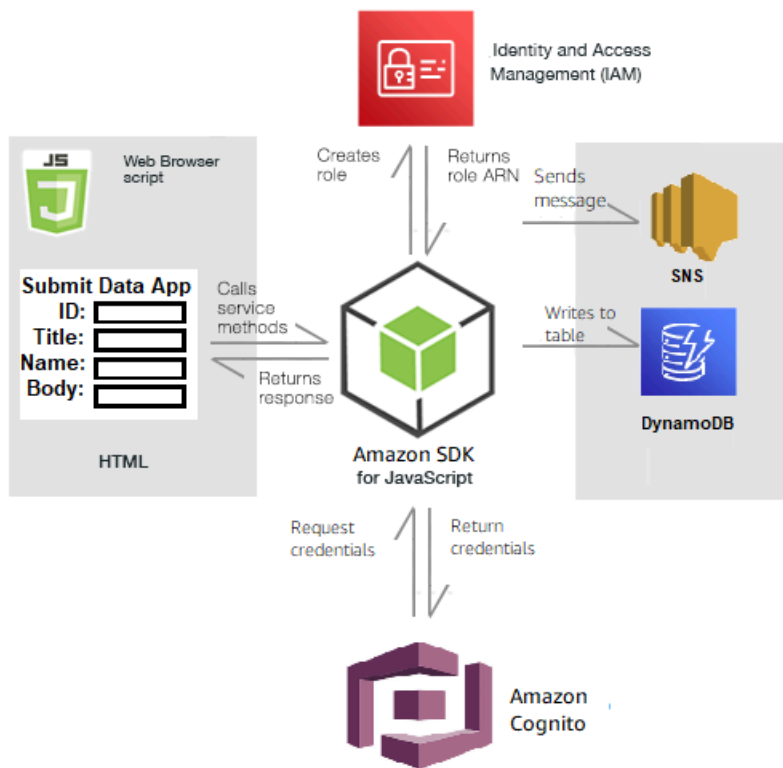
Crear una aplicación para enviar datos a DynamoDB

Este tutorial de Node.js para servicios cruzados muestra cómo crear una aplicación que permita a los usuarios enviar datos a una tabla de Amazon DynamoDB. Esta aplicación utiliza los siguientes servicios:

- AWS Identity and Access Management (IAM) y Amazon Cognito para autorizaciones y permisos.
- Amazon DynamoDB (DynamoDB) para crear y actualizar las tablas.
- Amazon Simple Notification Service (Amazon SNS) para notificar al administrador de la aplicación cuando un usuario actualiza la tabla.

El escenario

En este tutorial, una página HTML proporciona una aplicación basada en un navegador para enviar datos a una tabla de Amazon DynamoDB. La aplicación utiliza Amazon SNS para notificar al administrador de la aplicación cuando un usuario actualiza la tabla.



Para crear la aplicación:

1. [Requisitos previos](#)
2. [Aprovisionar recursos](#)
3. [Cree el HTML](#)
4. [Cree el script del navegador](#)
5. [Pasos siguientes](#)

Requisitos previos

Complete los siguientes requisitos previos:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos de AWS SDK for JavaScript y de terceros necesarios. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia las herramientas y los SDK de AWS.

Crear los recursos de AWS

Esta aplicación requiere los siguientes recursos:

- AWS Identity and Access Management (IAM), rol de usuario de Amazon Cognito no autenticado con los siguientes permisos:
 - sns:Publish
 - dynamodb:PutItem
- Una tabla de DynamoDB.

Puede crear estos recursos manualmente en la consola de AWS, pero le recomendamos que los aprovisiona usando AWS CloudFormation tal y como se describe en este tutorial.

Crear los recursos de AWS usando AWS CloudFormation

AWS CloudFormation le permite crear y aprovisionar implementaciones de infraestructura de AWS de forma predecible y uniforme. Para obtener más información sobre AWS CloudFormation, consulte la [Guía del usuario de AWS CloudFormation](#).

Para crear la pila de AWS CloudFormation usando AWS CLI:

1. Para instalar y configurar AWS CLI, siga las instrucciones de la [Guía del usuario de AWS CLI](#).
2. Cree un archivo con el nombre `setup.yaml` en el directorio raíz de la carpeta de su proyecto y copie en él [este contenido de GitHub](#).

Note

La plantilla de AWS CloudFormation se generó usando AWS CDK disponible [aquí en GitHub](#). Para obtener más información acerca del AWS CDK, consulte la [Guía para desarrolladores del AWS Cloud Development Kit \(CDK\)](#).

3. Ejecute el siguiente comando desde la línea de comandos y reemplace `STACK_NAME` por un nombre único para la pila, y `REGION` por su región de AWS.

⚠ Important

El nombre de la pila tiene que ser único dentro de una región de AWS y una cuenta de AWS. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

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

Para obtener más información sobre los parámetros de los comandos `create-stack`, consulte la [Guía referencia de comandos de AWS CLI](#) y la [Guía del usuario de AWS CloudFormation](#).

Para ver los recursos creados, abra AWS CloudFormation en la consola de administración de AWS, elija la pila y seleccione la pestaña Recursos.

4. Cuando se cree la pila, utilice AWS SDK for JavaScript para rellenar la tabla de DynamoDB, tal y como se describe en [Rellenar la tabla](#).

Rellenar la tabla

Para rellenar la tabla, primero cree un directorio llamado `libs` y, dentro de él, cree un archivo llamado `dynamoClient.js`, y pegue en él el contenido siguiente. Sustituya **REGION** por su región de AWS, y sustituya **IDENTITY_POOL_ID** por un ID de grupo de identidades de Amazon Cognito. Esto crea el objeto de cliente de DynamoDB.

```
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 DynamoDB service client object.
const dynamoClient = new DynamoDBClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION })
  })
});
```

```
    identityPoolId: IDENTITY_POOL_ID,
  }},
});

export { dynamoClient };
```

Este código está disponible [aquí en GitHub](#).

A continuación, cree una carpeta `dynamoAppHelperFiles` en la carpeta de su proyecto, cree un archivo `update-table.js` dentro de ella, y copie en él [este contenido de GitHub](#).

```
// 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();
```

En la línea de comandos, ejecute el comando siguiente:

```
node update-table.js
```

Este código está disponible [aquí en GitHub](#).

Crear una página front-end para la aplicación

Aquí se crea la página front-end HTML del navegador para la aplicación.

Cree un directorio `DynamoDBApp`, cree un archivo con el nombre `index.html` y copie en él [este código de GitHub](#). El elemento `script` añade el archivo `main.js`, que contiene todo el JavaScript necesario para el ejemplo. Creará el archivo `main.js` más adelante en este tutorial. El código restante de `index.html` crea la página del navegador que captura los datos que introducen los usuarios.

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Crear el script del navegador

En primer lugar, cree los objetos del cliente de servicio necesarios para el ejemplo. Cree un directorio `libs`, cree `snsClient.js`, y pegue el siguiente código en él. Sustituya *REGION* e *IDENTITY_POOL_ID* en cada uno.

Note

Utilice el ID del grupo de identidades de Amazon Cognito que creó en [Crear los recursos de 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 };
```

Este código está disponible [aquí en GitHub](#).

Para crear el script del navegador para este ejemplo, en una carpeta llamada DynamoDBApp, cree un módulo Node.js con el nombre de archivo `add_data.js` y pegue en él el código siguiente. La función `submitData` envía los datos a una tabla de DynamoDB y envía un mensaje de texto SMS al administrador de la aplicación mediante Amazon SNS.

En la función `submitData`, indique las variables para el número de teléfono objetivo, los valores introducidos en la interfaz de la aplicación y para el nombre del bucket de Amazon S3.

A continuación, cree un objeto de parámetros para añadir un elemento a la tabla. Si ninguno de los valores está vacío, `submitData` agrega el elemento a la tabla y envía el mensaje.

Recuerde establecer la función como disponible para del navegador con `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;
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Por último, ejecute lo siguiente en la línea de comandos para agrupar el JavaScript de este ejemplo en un archivo llamado `main.js`:

```
webpack add_data.js --mode development --target web --devtool false -o main.js
```

Note

Para obtener más información sobre cómo instalar Webpack, consulte [Combine aplicaciones con webpack](#).

Para ejecutar la aplicación, abra `index.html` en su navegador.

Eliminación de recursos

Como se indicó al principio de este tutorial, recuerde cancelar todos los recursos que cree mientras realiza este tutorial para asegurarse de que no se le cobre nada. Para ello, puede eliminar la pila de AWS CloudFormation que creó en el tema [Crear los recursos de AWS](#) de este tutorial, de la siguiente manera:

1. Abra [AWS CloudFormation en la consola de administración de AWS](#).
2. Abra la página Pilas y seleccione la pila.
3. Elija Eliminar.

Para ver más ejemplos de servicios cruzados de AWS, consulte [ejemplos de servicios cruzados de AWS SDK for JavaScript](#).

Crear una aplicación de transcripción con usuarios autenticados

En este tutorial, aprenderá a:

- Implemente la autenticación usando un grupo de identidades de Amazon Cognito para aceptar usuarios federados con un grupo de usuarios de Amazon Cognito.
- Utilice Amazon Transcribe para transcribir y mostrar grabaciones de voz en el navegador.

El escenario

La aplicación permite a los usuarios registrarse con un correo electrónico y un nombre de usuario únicos. Al confirmar su correo electrónico, pueden grabar mensajes de voz que se transcriben automáticamente y se muestran en la aplicación.

Cómo funciona

La aplicación utiliza dos buckets de Amazon S3, uno para alojar el código de la aplicación y otro para almacenar transcripciones. La aplicación utiliza un grupo de usuarios de Amazon Cognito para autenticar a los usuarios. Los usuarios autenticados tienen permisos de IAM para obtener acceso a los servicios de AWS requeridos.

La primera vez que un usuario graba un mensaje de voz, Amazon S3 crea una carpeta única con el nombre del usuario en el bucket de Amazon S3 para almacenar las transcripciones. Amazon Transcribe transcribe el mensaje de voz a texto y lo guarda en JSON en la carpeta del usuario. Cuando el usuario actualiza la aplicación, sus transcripciones se muestran y están disponibles para su descarga o eliminación.

Completar el tutorial debería tomarle aproximadamente 30 minutos.

Pasos

Para crear la aplicación:

1. [Requisitos previos](#)
2. [Crear los recursos de AWS](#)
3. [Cree el HTML](#)
4. [Preparar el script del navegador](#)
5. [Ejecutar la aplicación](#)
6. [Eliminar los recursos](#)

Requisitos previos

- Configure el entorno del proyecto para ejecutar los ejemplos de JavaScript de este nodo e instale los módulos necesarios de AWS SDK for JavaScript y de terceros. Siga las instrucciones en [GitHub](#).

- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de las herramientas y los SDK de AWS.

Important

Este ejemplo usa ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js..](#)

No obstante, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript.](#)

Crear los recursos de AWS

En esta sección se describe cómo aprovisionar recursos de AWS para esta aplicación utilizando el AWS Cloud Development Kit (AWS CDK).

Note

AWS CDK es un marco de desarrollo de software que le permite definir los recursos de las aplicaciones en la nube. Para obtener más información, consulte la [Guía para desarrolladores del AWS Cloud Development Kit \(CDK\).](#)

Para crear recursos para la aplicación, usa la plantilla disponible [aquí en GitHub](#) para crear una pila de AWS CDK usando la [Web Services Management Console de AWS](#) o [AWS CLI](#). Para obtener instrucciones sobre cómo modificar o eliminar la pila y sus recursos asociados cuando haya terminado el tutorial, consulta [GitHub](#).

Note

El nombre de la pila tiene que ser único dentro de una región de AWS y una cuenta de AWS. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

La pila resultante aprovisiona automáticamente los siguientes recursos.

- Un grupo de identidades de Amazon Cognito con un rol de usuario autenticado.
- El rol de usuario autenticado incluye una política de IAM con permisos para Amazon S3 y Amazon Transcribe.
- Un grupo de usuarios de Amazon Cognito que permite a los usuarios registrarse e iniciar sesión en la aplicación.
- Un bucket de Amazon S3 para alojar los archivos de la aplicación.
- Un bucket de Amazon S3 para almacenar las transcripciones.

Important

Este bucket de Amazon S3 permite el acceso público a READ (LIST), lo que permite a cualquier persona obtener una lista de los objetos del bucket y, potencialmente, hacer un mal uso de la información. Si no elimina este bucket de Amazon S3 inmediatamente después de completar el tutorial, le recomendamos encarecidamente que cumpla con las [prácticas de seguridad recomendadas en Amazon S3](#) de la Guía del usuario de Amazon Simple Storage Service.

Cree el HTML

Cree un archivo `index.html` y copie y pegue en él el contenido que aparece a continuación. La página incluye un panel de botones para grabar mensajes de voz y una tabla que muestra los mensajes transcritos anteriormente por el usuario actual. La etiqueta de script situada al final del elemento `body` invoca `main.js`, que contiene todos los scripts del navegador de la aplicación. El archivo `main.js` se crea mediante Webpack, tal y como se describe en la siguiente sección de este 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>
```

Este ejemplo de código está disponible [aquí en GitHub](#).

Prepare el script de navegador.

Hay tres archivos, `index.html`, `recorder.js` y `helper.js`, que tiene que agrupar en uno solo `main.js` usando Webpack. En esta sección se describen en detalle solo las funciones `index.js` que utilizan el SDK para JavaScript, que está disponible [aquí en GitHub](#).

Note

`recorder.js` y `helper.js` son obligatorios pero, debido a que no contienen el código Node.js, se explican en los comentarios en línea [aquí](#) y [aquí](#), respectivamente, en GitHub.

Primero, defina los parámetros. `COGNITO_ID` es el punto de conexión del grupo de usuarios de Amazon Cognito que creó en tema [Crear los recursos de AWS](#) de este tutorial. Su formato es `cognito-idp.AWS_REGION.amazonaws.com/USER_POOL_ID`. El ID del grupo de usuarios es `ID_TOKEN` en el token de credenciales de AWS, que la función `getToken` del archivo `helper.js` elimina de la URL de la aplicación. Este token se pasa a la variable `loginData`, que proporciona los inicios de sesión a los objetos de cliente de Amazon Transcribe y Amazon S3. Sustituya `"REGION"` por la región de AWS y `"BUCKET"` por `.`. Sustituya `"IDENTITY_POOL_ID"` por la `IdentityPoolId` de la página de muestra del grupo de identidades de Amazon Cognito que creó para este ejemplo. Esto también se transfiere a cada objeto de 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.
// 'COGNITO_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,
  }),
});
```

Cuando se carga la página HTML, `updateUserInterface` crea una carpeta con el nombre del usuario en el bucket de Amazon S3 si es la primera vez que inicia sesión en la aplicación. De lo contrario, actualiza la interfaz de usuario con las transcripciones de las sesiones anteriores del usuario.

```
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 yieldUnit8Chunks(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* yieldUnit8Chunks(data) {
```

```
const reader = data.getReader();
try {
  while (true) {
    const { done, value } = await reader.read();
    if (done) return;
    yield value;
  }
} finally {
  reader.releaseLock();
}
}
```

Cuando el usuario graba un mensaje de voz para su transcripción, `upload` carga las grabaciones en el bucket de Amazon S3. Esta función se llama desde el archivo `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` elimina una transcripción de la interfaz de usuario y `deleteRow` elimina una transcripción existente del bucket de Amazon S3. Ambas se activan mediante el botón Eliminar de la interfaz de usuario.

```
// 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 último, ejecute lo siguiente en la línea de comandos para agrupar el JavaScript de este ejemplo en un archivo llamado `main.js`:

```
webpack index.js --mode development --target web --devtool false -o main.js
```

Note

Para obtener más información sobre cómo instalar Webpack, consulte [Combine aplicaciones con webpack](#).

Ejecute la aplicación.

Puede ver la aplicación en la siguiente ubicación.

```
DOMAIN/login?  
client_id=APP_CLIENT_ID&response_type=token&scope=aws.cognito.signin.user.admin+email  
+openid+phone+profile&redirect_uri=REDIRECT_URL
```

Amazon Cognito facilita la ejecución de la aplicación al proporcionar un enlace en la Web Services Management Console de AWS. Solo tiene que ir a la configuración del cliente de aplicaciones de su grupo de usuarios de Amazon Cognito y seleccionar la IU alojada de Launch. La URL de la aplicación tiene el siguiente formato:

Important

La IU alojada tiene de forma predeterminada un tipo de respuesta de “código”. Sin embargo, este tutorial está diseñado para el tipo de respuesta “token”, por lo que debe cambiarlo.

Elimine los recursos de AWS

Cuando termine el tutorial, debe eliminar los recursos para no incurrir en cargos innecesarios. Como ha añadido contenido a ambos buckets de Amazon S3, tiene que eliminarlos manualmente. A continuación, puede eliminar los recursos restantes usando la [Web Services Management Console de AWS](#) o [AWS CLI](#). Consulta [aquí en GitHub](#) las instrucciones sobre cómo modificar o eliminar la pila y sus recursos asociados cuando haya terminado el tutorial.

Invocación de Lambda con API Gateway

Puede invocar una función de Lambda utilizando Amazon API Gateway, que es un servicio de AWS para la creación, la publicación, el mantenimiento, el monitoreo y la protección de las API REST, HTTP y de WebSocket a cualquier escala. Los desarrolladores de la API pueden crear API que obtengan acceso a AWS o a otros servicios web, así como los datos almacenados en la nube de

AWS. Como desarrollador de API Gateway, puede crear API para su uso en sus propias aplicaciones de cliente. Para obtener más información, consulte [Qué es Amazon API Gateway](#)

Lambda es un servicio de computación que permite ejecutar código sin aprovisionar ni administrar servidores. Puede crear funciones de Lambda en varios lenguajes de programación. Para obtener más información acerca de Lambda, consulte [Qué es Lambda](#).

En este ejemplo, creará una función de Lambda utilizando la API de tiempo de ejecución de Lambda JavaScript. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. Por ejemplo, supongamos que una organización envía un mensaje de texto al móvil a sus empleados para felicitarles por su primer aniversario, como se muestra en esta ilustración.



Completar el ejemplo debería tomarle aproximadamente 20 minutos.

En este ejemplo, se muestra cómo utilizar la lógica de JavaScript para crear una solución que ejecute este caso de uso. Por ejemplo, aprenderá a leer una base de datos para determinar qué empleados han cumplido un año, cómo procesar los datos y cómo enviar un mensaje de texto, todo ello utilizando una función de Lambda. A continuación, aprenderá a usar API Gateway para invocar esta función de Lambda utilizando un punto de conexión de Rest. Por ejemplo, puede invocar la función de Lambda con este comando de curl:

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/employee"
```

En este tutorial de AWS se utiliza una tabla de Amazon DynamoDB denominada Employee que contiene estos campos.

- id: la clave principal de la tabla.
- firstName: nombre del empleado.
- teléfono: número de teléfono del empleado.
- startDate: fecha de inicio del empleado.

<input type="checkbox"/>	Id ⓘ	first	phone	startDate
<input type="checkbox"/>	1	Scott	15555555654	2019-12-20
<input type="checkbox"/>	2	Malcolm	15555555654	2019-12-17
<input type="checkbox"/>	55	Lam	15555555654	2019-12-19

Important

Costo de finalización: los servicios de AWS incluidos en este documento están incluidos en el nivel gratuito de AWS. Sin embargo, asegúrese de cancelar todos los recursos una vez que haya completado este ejemplo para que no se le cobre nada.

Para crear la aplicación:

1. [Completar los requisitos previos](#)
2. [Crear los recursos de AWS](#)
3. [Preparar el script del navegador](#)
4. [Crear y cargar una función de Lambda](#)
5. [Implementar la función de Lambda](#)
6. [Ejecutar la aplicación](#)
7. [Eliminar los recursos](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos necesarios de AWS SDK for JavaScript y de terceros. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre proporcionar un archivo de credenciales compartido, consulte [Archivos de](#)

[configuración y credenciales compartidos](#) en la Guía de referencia las herramientas y los SDK de AWS.

Crear los recursos de AWS

Este tutorial requiere los siguientes recursos:

- Una tabla de Amazon DynamoDB llamada Employee con una clave llamada Id y los campos que se muestran en la ilustración anterior. Asegúrese de introducir los datos correctos, incluido un teléfono móvil válido con el que quiera probar este caso de uso. Para obtener más información, consulte [Crear una tabla](#).
- Un rol de IAM con permisos adjuntos para ejecutar funciones de Lambda.
- Un bucket de Amazon S3 para alojar la función de Lambda.

Puede crear estos recursos manualmente, pero le recomendamos que los aprovisiona utilizando AWS CloudFormation tal y cómo se describe en este tutorial.

Crear los recursos de AWS usando AWS CloudFormation

AWS CloudFormation le permite crear y aprovisionar implementaciones de infraestructura de AWS de forma predecible y uniforme. Para obtener más información sobre AWS CloudFormation, consulte la [Guía del usuario de AWS CloudFormation](#).

Para crear la pila de AWS CloudFormation usando AWS CLI:

1. Para instalar y configurar AWS CLI, siga las instrucciones de la [Guía del usuario de AWS CLI](#).
2. Cree un archivo con el nombre `setup.yaml` en el directorio raíz de la carpeta de su proyecto y copie en él [este contenido de GitHub](#).

Note

La plantilla de AWS CloudFormation se generó usando AWS CDK disponible [aquí en GitHub](#). Para obtener más información acerca de AWS CDK, consulte la [Guía para desarrolladores de AWS Cloud Development Kit \(CDK\)](#).

3. Ejecute el siguiente comando desde la línea de comandos y reemplace `STACK_NAME` por un nombre único para la pila.

⚠ Important

El nombre de la pila tiene que ser único en una región de AWS de una cuenta de AWS. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

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

Para obtener más información sobre los parámetros del comando `create-stack`, consulte la [guía de referencia de comandos de AWS CLI](#) y la [Guía del usuario de AWS CloudFormation](#).

4. A continuación, complete la tabla siguiendo el procedimiento [Rellenar la tabla](#).

Rellenar la tabla

Para rellenar la tabla, primero cree un directorio con el nombre `libs` y, dentro de él, cree un archivo con el nombre `dynamoClient.js` y pegue en él el contenido que aparece a continuación.

```
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 };
```

Este código está disponible [aquí en GitHub](#).

A continuación, cree un archivo con el nombre `populate-table.js` en el directorio raíz de la carpeta de su proyecto y copie en él [este contenido de GitHub](#). Para uno de los elementos, sustituya el valor de la propiedad `phone` por un número de teléfono móvil válido con el formato E.164, y el valor de `startDate` por la fecha de hoy.

Desde la línea de comandos, ejecute el comando siguiente:

```
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: "1555555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "1555555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "1555555555555652" },
            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();
```

Este código está disponible [aquí en GitHub](#).

Creación de la función de Lambda

Configuración del SDK

En el directorio `libs`, cree archivos con los nombres `snsClient.js` y `lambdaClient.js`, y pegue el contenido que aparece a continuación en estos archivos, 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 };
```

Sustituya **REGION** por la región de AWS. Este código está disponible [aquí en 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 };
```

Sustituya **REGION** por la región de AWS. Este código está disponible [aquí en GitHub](#).

En primer lugar, importe los módulos y comandos necesarios de AWS SDK for JavaScript (v3). A continuación, calcule la fecha de hoy y asígnela a un parámetro. En tercer lugar, cree los parámetros para `ScanCommand`. Sustituya **TABLE_NAME** por el nombre de la tabla que creó en la sección [Crear los recursos de AWS](#) de este ejemplo.

El siguiente fragmento de código muestra este paso. (Consulte [Agrupación de la función de Lambda para ver el ejemplo 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",
};
```

Escaneo de una tabla de DynamoDB

En primer lugar, cree una función `async/await` llamada `sendText` para publicar un mensaje de texto usando `PublishCommand` de Amazon SNS. A continuación, añada un patrón de bloques `try` que escanee la tabla de DynamoDB en busca de empleados cuyo aniversario laboral sea hoy y, a continuación, llame a la función `sendText` para enviar un mensaje de texto a estos empleados. Si se produce un error, se llama al bloque `catch`.

El siguiente fragmento de código muestra este paso. (Consulte [Agrupación de la función de Lambda para ver el ejemplo 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);
  }
};
```

Agrupación de la función de Lambda

En este tema se describe cómo agrupar `mylambdafunction.ts` y los módulos AWS SDK for JavaScript necesarios para este ejemplo en un archivo de agrupación llamado `index.js`.

1. Si aún no lo ha hecho, siga el [Tareas previas necesarias](#) para este ejemplo para instalar el paquete web.

Note

Para obtener información sobre el paquete web, consulte [Combine aplicaciones con webpack](#).

2. Ejecute lo siguiente en la línea de comandos para agrupar el JavaScript de este ejemplo en un archivo llamado `<index.js>`:

```
webpack mylambdafunction.ts --mode development --target node --devtool false --output-library-target umd -o index.js
```

⚠ Important

Observe que la salida tiene el nombre `index.js`. Esto se debe a que las funciones de Lambda tienen que tener un controlador `index.js` para funcionar.

3. Comprima el archivo de salida empaquetado, `index.js`, en un archivo ZIP denominado `mylambdafunction.zip`.
4. Suba `mylambdafunction.zip` al bucket de Amazon S3 que creó en el tema de [Crear los recursos de AWS](#) de este tutorial.

Implemente la función de Lambda

En la raíz del proyecto, cree un archivo `lambda-function-setup.ts` y pegue en él el contenido siguiente.

Sustituya ***BUCKET_NAME*** por el nombre del bucket de Amazon S3 en el que cargó la versión ZIP de la función de Lambda. Sustituya ***ZIP_FILE_NAME*** por el nombre de la versión ZIP de la función de Lambda. Sustituya ***ROLE*** por el número de recurso de Amazon (ARN) del rol de IAM que ha creado en el tema [Crear los recursos de AWS](#) de este tutorial. Sustituya ***LAMBDA_FUNCTION_NAME*** por un nombre para la función de 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();
```

Introduzca lo siguiente en la línea de comandos para implementar la función de Lambda.

```
node lambda-function-setup.ts
```

Este ejemplo de código está disponible [aquí en GitHub](#).

Configuración de API Gateway para invocar la función de Lambda

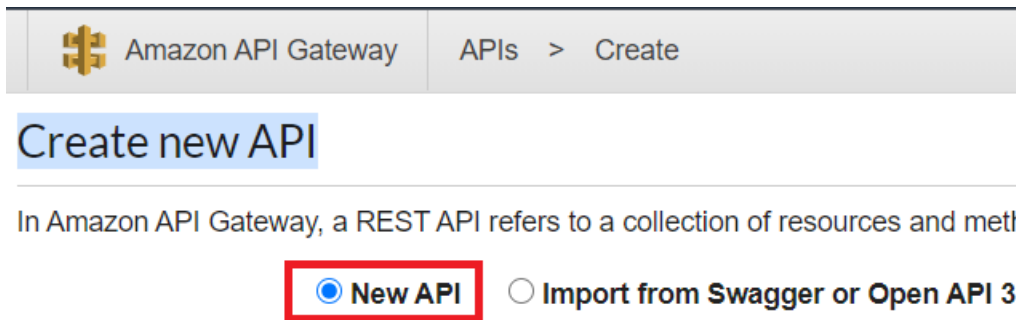
Para crear la app:

1. [Cree el resto de la API](#)
2. [Probar el método de API Gateway](#)
3. [Implementar el método de API Gateway](#)

Cree el resto de la API

Puede utilizar la consola de API Gateway para crear un punto de conexión para la función de Lambda. Una vez hecho esto, podrá invocar la función de Lambda mediante una llamada restful.

1. Inicie sesión en la consola de [Amazon API Gateway](#).
2. En Rest API, seleccione Compilación.
3. Seleccione Nueva API.



Amazon API Gateway APIs > Create

Create new API


In Amazon API Gateway, a REST API refers to a collection of resources and meth

New API Import from Swagger or Open API 3

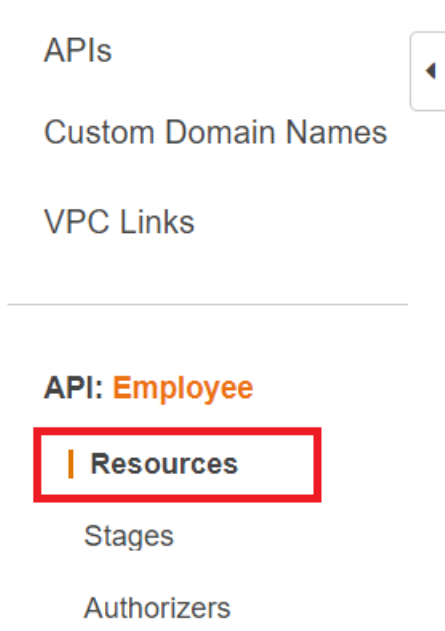
4. Especifique Empleado como nombre de la API y proporcione una descripción.

Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="Employee"/>
Description	<input type="text" value="This invokes a Lambda function"/>
Endpoint Type	<input type="text" value="Regional"/> 

5. Elija Crear API.
6. Seleccione Recursos en la sección Empleado.



7. En el campo de nombre, especifique los empleados.
8. Elija Create Resources (Crear recursos).
9. En el menú desplegable Acciones, elija Crear recursos.

Use this page to create a new child resource for your resource. 📌

Configure as [proxy resource](#)

ⓘ

Resource Name*

employees

Resource Path*

/ employees

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/{proxy+}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/foo`. To handle requests to `/`, add a new ANY method on the `/` resource.

Enable API Gateway CORS

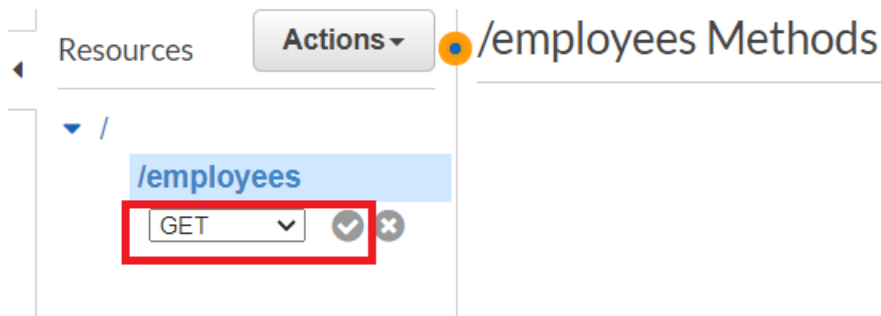
ⓘ

* Required

Cancel

Create Resource

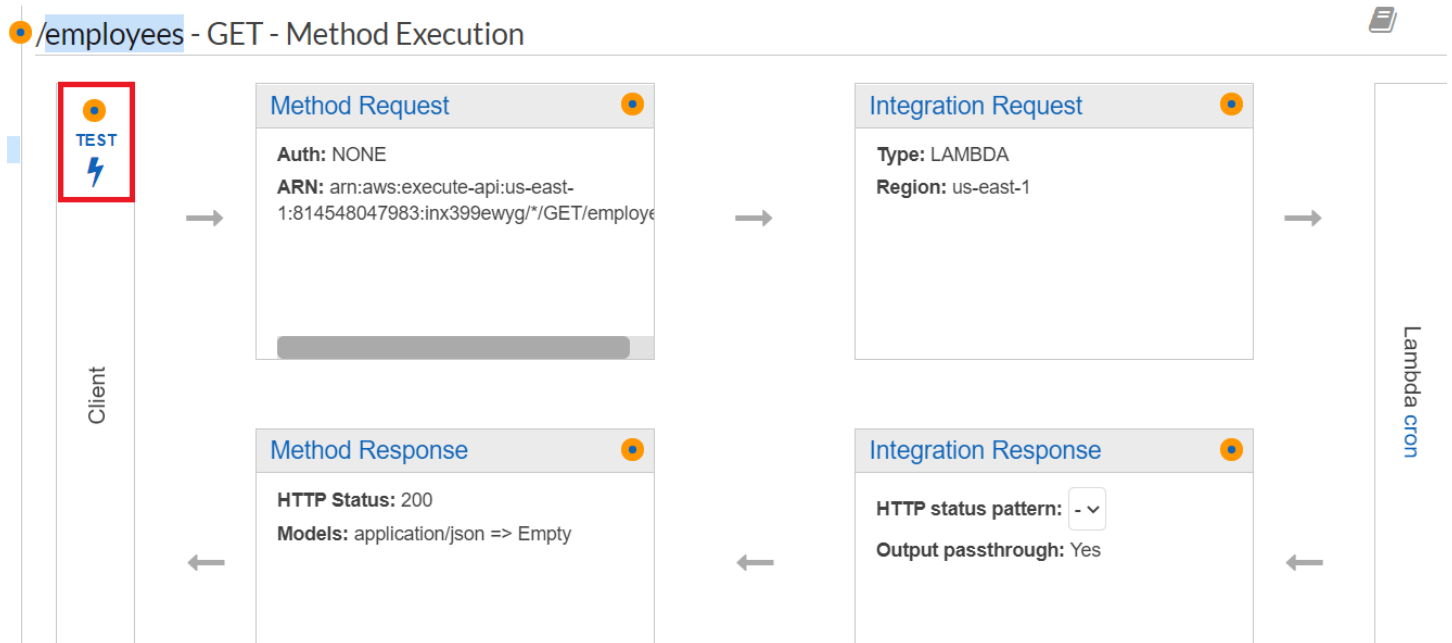
10. Elija `/empleados`, seleccione Crear método en las Acciones y, a continuación, seleccione OBTENER en el menú desplegable situado debajo de `/empleados`. Elija el icono de marca de verificación.



11. Elija la función de Lambda e introduzca mylambdafunction como nombre de la función de Lambda. Seleccione Guardar.

Probar el método de API Gateway

En este punto del tutorial, puede probar el método de API Gateway que invoca la función de Lambda mylambdafunction. Para probar el método, elija Probar, como se muestra en la siguiente ilustración.

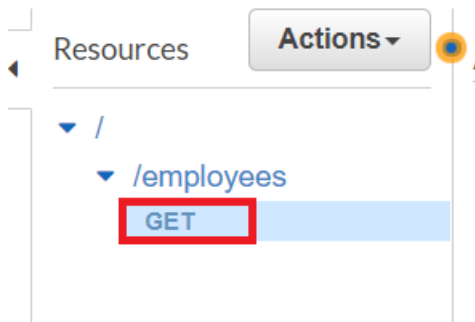


Una vez que se invoca la función de Lambda, puede ver el archivo de registro para ver un mensaje correcto.

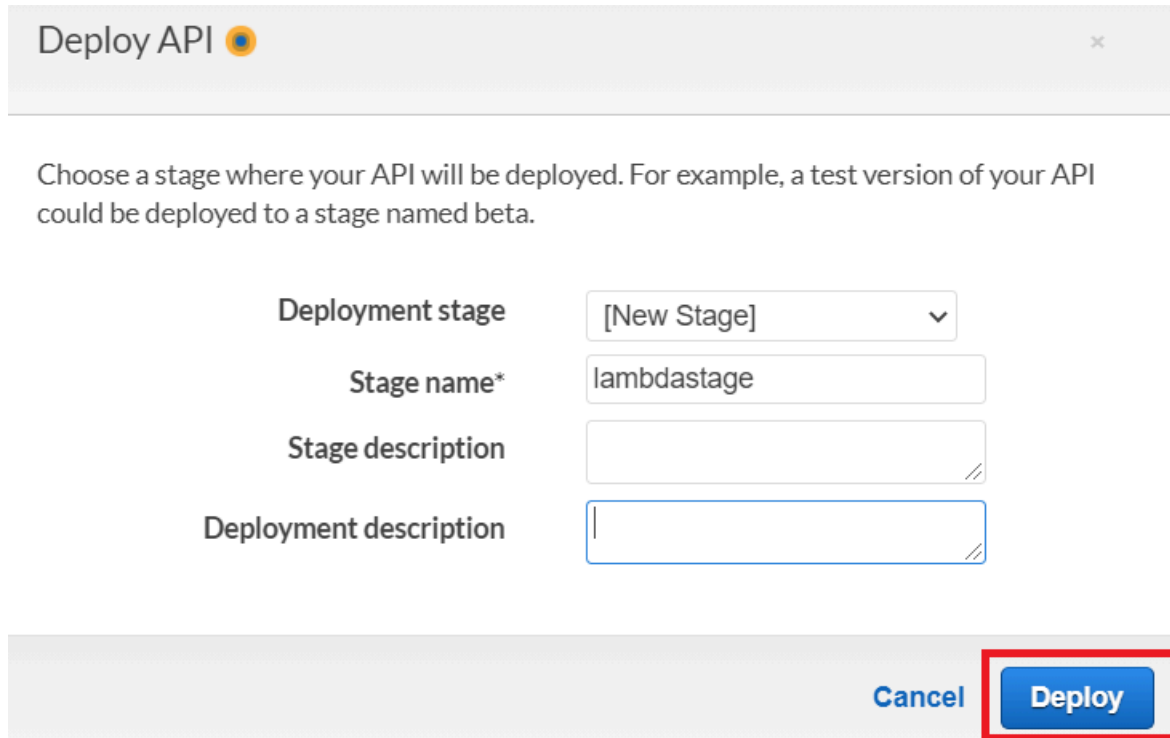
Implementar el método de API Gateway

Una vez que la prueba se haya realizado correctamente, podrá implementar el método desde la [consola de Amazon API Gateway](#).

1. Elija Obtener.



2. En el menú desplegable Acciones, elija Implementar API.

A screenshot of the 'Deploy API' dialog box. The title bar says 'Deploy API' with a yellow dot and a close button. Below the title bar, there is a paragraph of text: 'Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.' Below this text, there are four form fields: 'Deployment stage' with a dropdown menu showing '[New Stage]'; 'Stage name*' with a text input field containing 'lambdastage'; 'Stage description' with an empty text area; and 'Deployment description' with an empty text area. At the bottom right of the dialog, there are two buttons: 'Cancel' and 'Deploy'. The 'Deploy' button is highlighted with a red rectangular box.

3. Rellene el formulario de Implementar API y seleccione Implementar.

Deploy API ✕

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	[New Stage] ▾
Stage name*	lambdastage
Stage description	<input type="text"/>
Deployment description	<input type="text"/>

Cancel Deploy

4. Elija Guardar cambios.
5. Seleccione Obtener de nuevo y observe que la URL cambia. Esta es la URL de invocación que puede utilizar para invocar la función de Lambda.

Stages Create lambdastage - GET - /employees

- lambdastage
 - /employees
 - GET

Invoke URL: <https://...ewyg.execute-api.us-east-1.amazonaws.com/lambdastage/employees>

Use this page to override the `lambdastage` stage settings for the GET to /employees method.

Settings Inherit from stage Override for this method

Elimine los recursos

¡Enhorabuena! Ha invocado una función de Lambda a través de Amazon API Gateway usando AWS SDK for JavaScript. Como se indicó al principio de este tutorial, asegúrese de cancelar todos los recursos que cree mientras realiza este tutorial para asegurarse de que no se le cobre nada. Para ello, puede eliminar la pila de AWS CloudFormation que creó en el tema [Crear los recursos de AWS](#) de este tutorial, de la siguiente manera:

1. Abra [AWS CloudFormation en la consola de administración de AWS](#).

2. Abra la página Pilas y seleccione la pila.
3. Elija Eliminar.

Creación de flujos de trabajo de AWS sin servidor usando AWS SDK for JavaScript

Puede crear un flujo de trabajo de AWS sin servidor mediante Step Functions, el SDK for Java de AWS y AWS Step Functions. Cada paso del flujo de trabajo se implementa con una función de Lambda. Lambda es un servicio de computación que permite ejecutar código sin aprovisionar ni administrar servidores. Step Functions es un servicio de orquestación sin servidor que le permite combinar funciones de Lambda y otros servicios de AWS para crear aplicaciones esenciales desde el punto de vista empresarial.

Note

Puede crear funciones de Lambda en varios lenguajes de programación. Para este tutorial, se implementaron funciones de Lambda mediante la API Java de Lambda. Para obtener más información sobre Lambda, consulte [Qué es Lambda](#)

En este tutorial creará un flujo de trabajo que crea tickets de soporte para una organización. Cada paso del flujo de trabajo realiza una operación en el ticket. En este tutorial se muestra cómo utilizar JavaScript para procesar datos de flujo de trabajo. Por ejemplo, aprenderá a leer los datos que se pasan al flujo de trabajo, a pasar datos entre pasos y a invocar servicios de AWS desde el flujo de trabajo.

Costo de finalización: los servicios de AWS incluidos en este documento están incluidos en la [suscripción gratuita de AWS](#).

Nota: Recuerde cancelar todos los recursos que cree mientras sigue este tutorial para asegurarse de que no se le cobre nada.

Temas

- [Tareas previas necesarias](#)
- [Crear los recursos de AWS](#)
- [Creación de un flujo de trabajo](#)
- [Crear la función de Lambda](#)

- [Añadir funciones de Lambda a flujos de trabajo](#)
- [Ejecute su flujo de trabajo mediante la consola de Step Functions](#)
- [Elimine los recursos de AWS](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos ejemplos de Node TypeScript e instale los módulos necesarios de AWS SDK for JavaScript y de terceros. Siga las instrucciones en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia las herramientas y los SDK de AWS.

Crear los recursos de AWS

Este tutorial requiere los siguientes recursos.

- Una tabla de Amazon DynamoDB llamada Case con una clave llamada Id.
- Un nombre de rol de IAM llamado `lambda-support` utilizado para invocar funciones de Lambda. Este rol tiene políticas que le permiten invocar los servicios de Amazon DynamoDB y Amazon Simple Email Service desde una función de Lambda.
- Un nombre de rol de IAM llamado `workflow-support` utilizado para invocar el flujo de trabajo.
- Un bucket de Amazon S3 para alojar las funciones de Lambda.


Puede crear estos recursos manualmente, pero le recomendamos que los aprovisiones utilizando el AWS Cloud Development Kit (CDK) (AWS CDK), tal y cómo se describe en este tutorial.

Cree los recursos de AWS usando AWS CloudFormation

AWS CloudFormation le permite crear y aprovisionar implementaciones de infraestructura de AWS de forma predecible y uniforme. Para obtener más información sobre AWS CloudFormation, consulte la [Guía del usuario de AWS CloudFormation](#).


Para crear la pila de AWS CloudFormation:

1. Para instalar y configurar AWS CLI, siga las instrucciones de la [Guía del usuario de AWS CLI](#).
2. Cree un archivo con el nombre `setup.yaml` en el directorio raíz de la carpeta de su proyecto y copie en él [este contenido de GitHub](#).

 Note

La plantilla de AWS CloudFormation se generó usando AWS CDK disponible [aquí en GitHub](#). Para obtener más información acerca de AWS CDK, consulte la [Guía para desarrolladores de AWS Cloud Development Kit \(CDK\)](#).

3. Ejecute el siguiente comando desde la línea de comandos y reemplace `STACK_NAME` por un nombre único para la pila.

 Important


El nombre de la pila tiene que ser único en una región de AWS de una cuenta de AWS. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

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

Para obtener más información sobre los parámetros de los comandos `create-stack`, consulte la [Guía de referencia de comandos de AWS CLI](#) y la [Guía del usuario de AWS CloudFormation](#).

Cree los recursos de AWS usando la consola de administración de Amazon Web Services;

Para crear recursos para la aplicación en la consola, siga las instrucciones de la [Guía del usuario de AWS CloudFormation](#). Use la plantilla proporcionada, cree un archivo llamado `setup.yaml` y copie en él [este contenido de GitHub](#).

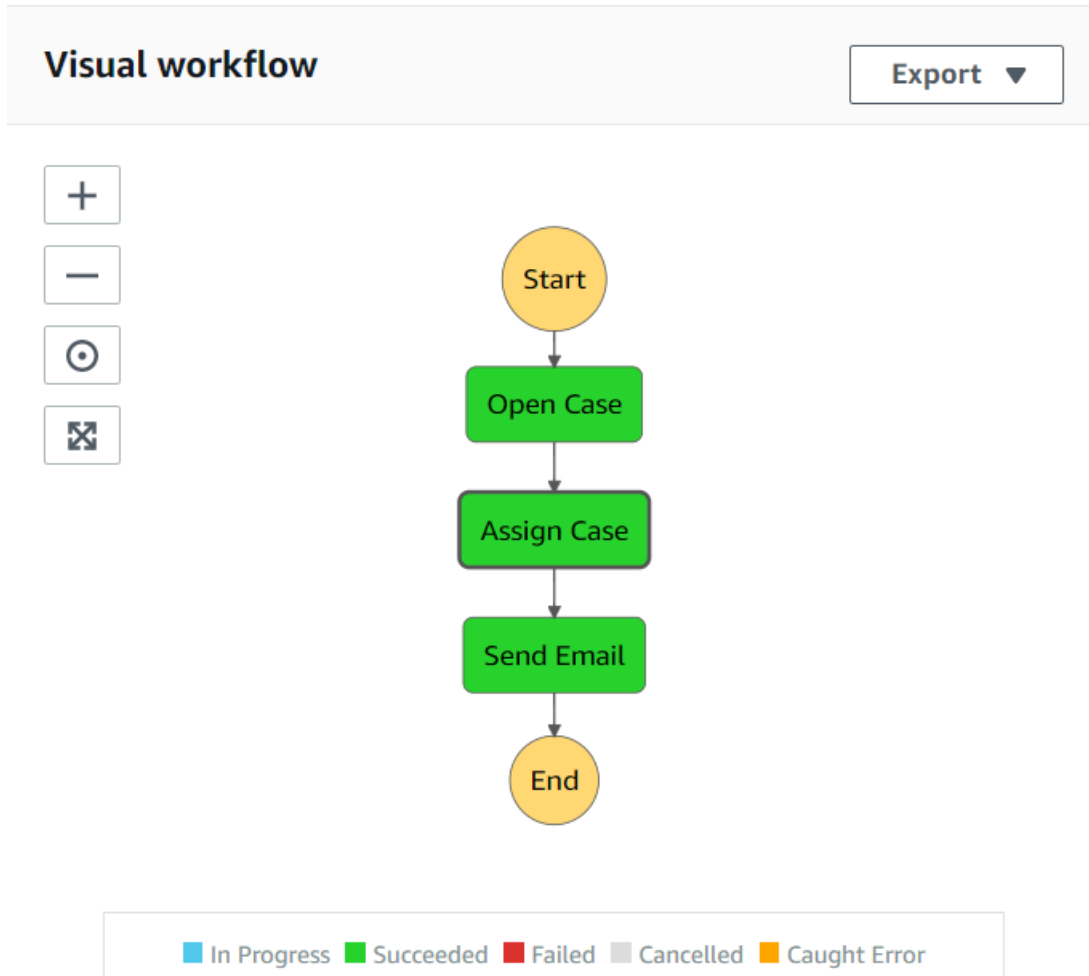
 Important

El nombre tiene que ser único dentro de una región de AWS y una cuenta de AWS. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

Para ver una lista de los recursos de la consola, abra la pila en el panel de control de AWS CloudFormation y seleccione la pestaña Recursos. Los necesitará para el tutorial.

Creación de un flujo de trabajo

En la siguiente imagen se muestra el flujo de trabajo que creará con este tutorial.



Lo siguiente es lo que ocurre en cada paso del flujo de trabajo:

- + Inicio: inicia el flujo de trabajo.
- + Abrir caso: gestiona el valor del ID de un ticket de soporte pasándolo al flujo de trabajo.
- + Asignar caso: asigna el caso de soporte a un empleado y almacena los datos en una tabla de DynamoDB.
- + Enviar correo electrónico: envía al empleado un mensaje de correo electrónico mediante Amazon Simple Email Service (Amazon SES) para informarle de que hay un nuevo ticket.

+ Finalizar: detiene el flujo de trabajo.

Crear un flujo de trabajo sin servidor usando Step Functions

Puede crear un flujo de trabajo que procese los tickets de soporte. Para definir un flujo de trabajo mediante Step Functions, debe crear un documento en Amazon States Language (basado en JSON) para definir su máquina de estados. Un documento de Amazon States Language describe cada paso. Tras definir el documento, Step Functions proporciona una representación visual del flujo de trabajo. La siguiente figura muestra el documento de Amazon States Language y la representación visual del flujo de trabajo.

Los flujos de trabajo pueden transferir datos entre pasos. Por ejemplo, el paso Abrir caso procesa un valor de identificador de caso (que se pasa al flujo de trabajo) y pasa ese valor al paso Asignar caso. Más adelante en este tutorial, creará una lógica de aplicación en la función de Lambda para leer y procesar los valores de los datos.

Para crear un flujo de trabajo

1. Abra [Amazon Web Services Console](#).
2. Seleccione Create State Machine.
3. Seleccione Author with code snippets (Autor con fragmentos de código). En el área Tipo, elija Estándar.

The screenshot shows the 'Define state machine' interface in the AWS console. It features three main options for authoring the state machine:

- Author with code snippets** (Selected): Author your workflow using Amazon States Language. You can generate code snippets to easily build out your workflow steps.
- Run a sample project**: Deploy and run a fully functioning sample project in minutes using CloudFormation.
- Start with a template**: Get started quickly with common patterns for Amazon States Language.

Below these options is a 'Type' section with two choices:

- Standard** (Selected): Durable, checkpointed workflows for machine learning, order fulfillment, IT/DevOps automation, ETL jobs, and other long-duration workloads.
- Express** (New): Event-driven workflows for streaming data processing, microservices orchestration, IoT data ingestion, mobile backends, and other short duration, high-event-rate workloads.

A 'Help me decide' link is located at the bottom left of the 'Type' section.

4. Introduzca el siguiente código para especificar el documento de Amazon States Language.

```
{
```

```
"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

No se preocupe por los errores relacionados con los valores de los recursos de Lambda. Actualizará estos valores más adelante en este tutorial.

5. Elija Siguiente.
6. En el campo del nombre, escriba SupportStateMachine.
7. En la sección Permisos, elija Elegir un rol existentes.
8. Elija workflow-support (el rol de IAM que creó).

Permissions

Execution role
The IAM role that defines which resources your state machine has permission to access during execution. To create a custom role, go to the [IAM console](#).

Create new role
Let Step Functions create a new role for you based on your state machine's definition and configuration details.

Choose an existing role

Enter a role ARN

Existing roles

workflow-support ▼ ↻

9. Elija Crear máquina de estado. Aparecerá un mensaje que indicará que la máquina de estados se creó correctamente.

✔ State machine successfully created ✕

Step Functions > State machines > SupportStateMachine

SupportStateMachine

Edit Start execution Delete Actions ▼

Details

ARN	arn:aws:states:us-west-2:81...:stateMachine:SupportStateMachine	Type	Standard
IAM role ARN	arn:aws:iam::81454...role/service-role/StepFunctions-SupportStateMachine-role	Creation date	Jun 1, 2020 02:53:16.855 PM

Executions | Logging | Definition | Tags

Crear la función de Lambda

Utilice la API de tiempo de ejecución de Lambda para crear las funciones de Lambda. En este ejemplo, hay tres pasos del flujo de trabajo, cada uno de los cuales corresponde a cada función de Lambda.

Cree estas funciones de Lambda, tal y como se describe en las siguientes secciones:

- [Función getId de Lambda](#): se utiliza como primer paso del flujo de trabajo que procesa el valor del ID del ticket.
- [Clase addItem de Lambda](#): se utiliza como segundo paso del flujo de trabajo que asigna el ticket a un empleado y almacena los datos en una base de datos de DynamoDB.

- [Clase sendemail de Lambda](#): se utiliza como el tercer paso del flujo de trabajo en el que se utiliza Amazon SES para enviar un mensaje de correo electrónico al empleado para informarle del ticket.

Función getId de Lambda

Cree una función de Lambda que devuelva el valor del ID del ticket que se pasa al segundo paso del flujo de trabajo.

```
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);
}
};
```

Introduzca lo siguiente en la línea de comandos para usar Webpack para agrupar el archivo en un archivo llamado `index.js`.

```
webpack getId.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

A continuación, comprima `index.js` en un archivo ZIP con el nombre `getId.js.zip`. Cargue el archivo ZIP en el bucket de Amazon S3 que creó en el tema de este ejemplo.

Este ejemplo de código está disponible [aquí en GitHub](#).

Clase addItem de Lambda

Cree una función de Lambda que seleccione a un empleado para asignarle el ticket y, a continuación, almacene los datos del ticket en una tabla de DynamoDB denominada `Case`.

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

Introduzca lo siguiente en la línea de comandos para usar Webpack para agrupar el archivo en un archivo llamado `index.js`.

```
webpack additem.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

A continuación, comprima `index.js` en un archivo ZIP con el nombre `additem.js.zip`. Cargue el archivo ZIP en el bucket de Amazon S3 que creó en el tema de este ejemplo.

Este ejemplo de código está disponible [aquí en GitHub](#).

Clase sendemail de Lambda

Cree una función de Lambda que envíe un correo electrónico para notificarle sobre el nuevo ticket. Se utiliza la dirección de correo electrónico que se transfiere en el segundo paso.

```
// 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 receipient = 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 = `<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);
}
};
```

Introduzca lo siguiente en la línea de comandos para usar Webpack para agrupar el archivo en un archivo llamado `index.js`.

```
webpack sendemail.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

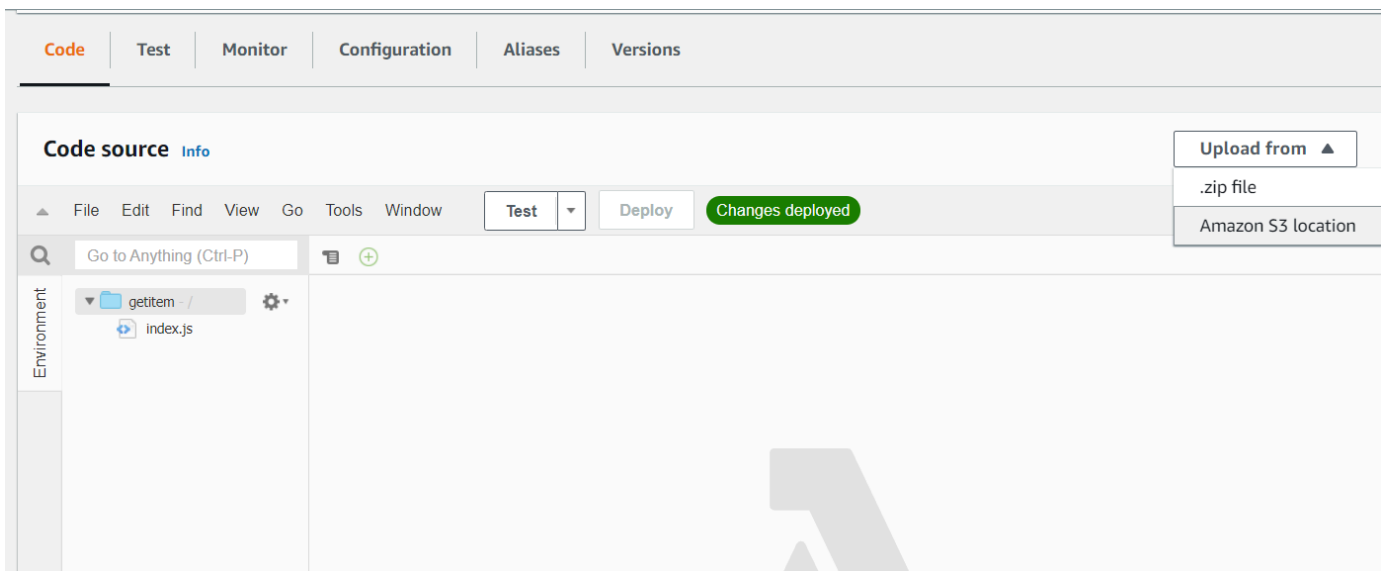
A continuación, comprima `index.js` en un archivo ZIP con el nombre `sendemail.js.zip`. Cargue el archivo ZIP en el bucket de Amazon S3 que creó en el tema de este ejemplo.

Este ejemplo de código está disponible [aquí en GitHub](#).

Implementar funciones de Lambda

Para implementar la función de Lambda `getid`:

1. Abra la consola de Lambda en [Amazon Web Services Console](#).
2. Elija Crear función.
3. Elija Crear desde cero.
4. En la sección de información básica, introduzca `getid` como nombre.
5. En Tiempo de ejecución, elija Node.js 14x.
6. Elija Usar un rol existente y, a continuación, elija `lambda-support` (el rol de IAM que creó en la).
7. Elija Crear función.
8. seleccione Cargar desde: ubicación de Amazon S3.
9. Seleccione Cargar, seleccione Cargar desde: ubicación de Amazon S3 e introduzca la URL del enlace de Amazon S3.

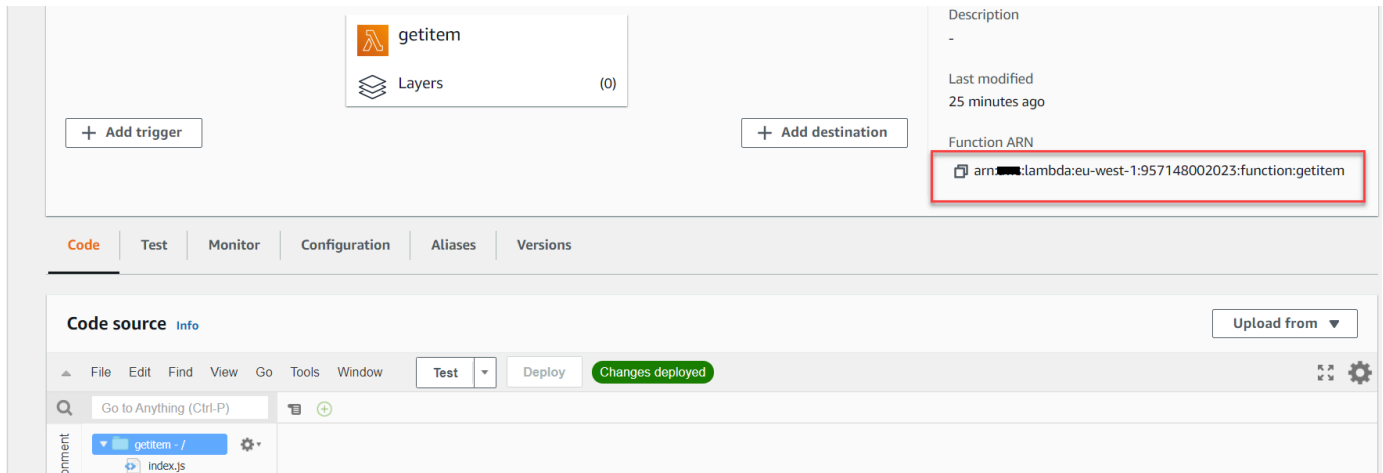


10. Seleccione Guardar.

11. Repita este procedimiento para `additem.js.zip` y `sendemail.js.zip` en las nuevas funciones de Lambda. Cuando termine, dispondrá de tres funciones de Lambda a las que podrá hacer referencia en el documento de Amazon States Language.

Añadir funciones de Lambda a flujos de trabajo

1. Abra la consola de Lambda. Observe que puede ver el valor del Nombre de recurso de Amazon (ARN) de Lambda en la esquina superior derecha.



2. Copie el valor y péguelo en el paso 1 del documento de Amazon States Language, ubicado en la consola de Step Functions.
3. Actualice el recurso para los pasos Asignar caso y Enviar correo electrónico. Así es como se enlazan las funciones de Lambda creadas mediante el SDK para Java de AWS a un flujo de trabajo creado con Step Functions.

Ejecute su flujo de trabajo mediante la consola de Step Functions

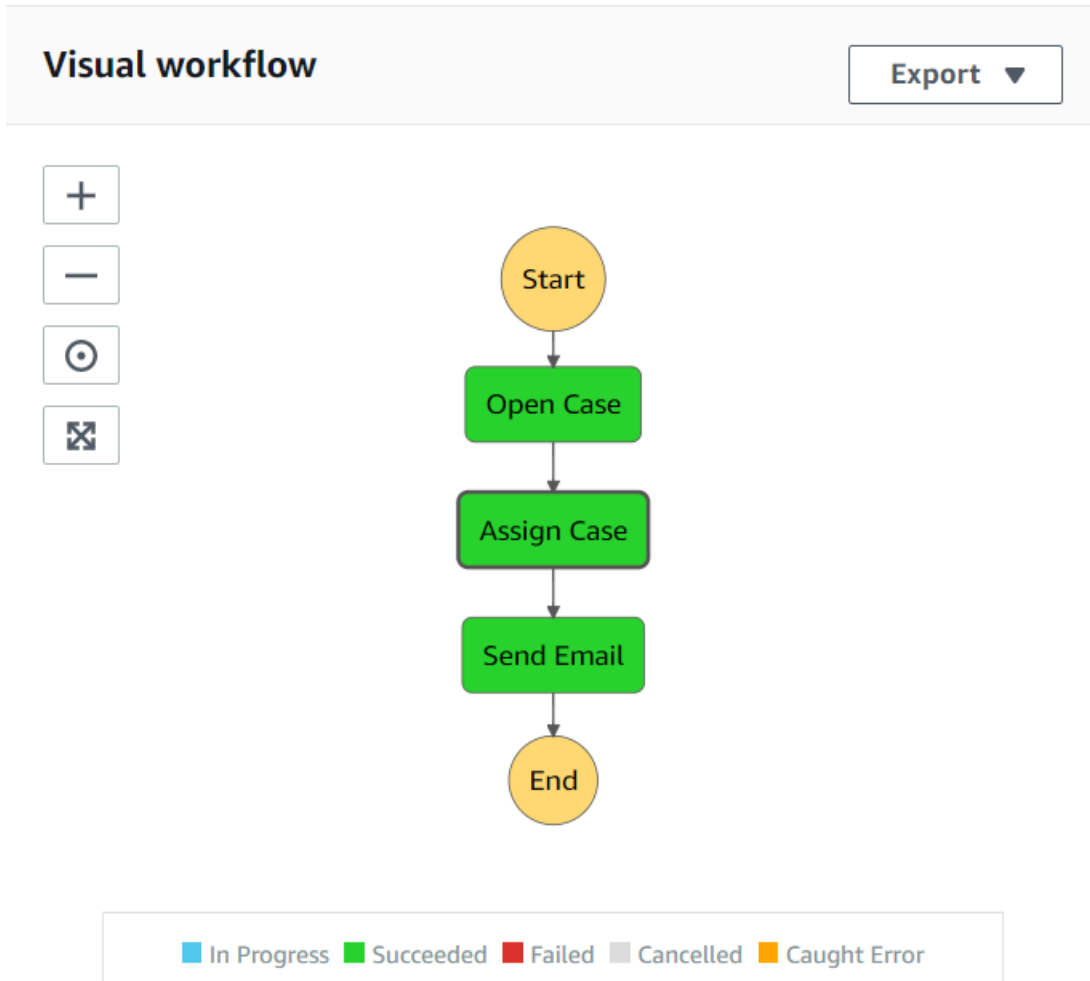
Puede invocar el flujo de trabajo en la consola de Step Functions. Una ejecución recibe una entrada JSON. Para este ejemplo, puede pasar los siguientes datos de JSON al flujo de trabajo.

```
{
  "inputCaseID": "001"
}
```

Para ejecutar su flujo de trabajo:

1. En la consola de Step Functions, seleccione Iniciar ejecución.

2. En la sección Entrada, pase los datos de JSON. Observe el flujo de trabajo. A medida que se vaya completando cada paso, se pondrán en verde.



3. Si el paso se pone en rojo, se ha producido un error. Puede hacer clic en el paso y ver los registros a los que se puede acceder desde el lado derecho.

Code | **Step details**

Name	Type
Assign Case	Task

Status

✔ Succeeded

Resource

arn:aws:lambda:us-west-2:8145-... :function:function3 [CloudWatch logs](#)

▶ Input

▶ Output

▶ Exception

Cuando finalice el flujo de trabajo, podrá ver los datos en la tabla de DynamoDB.

Scan: [Table] Case: id ^

Scan [Table] Case: id ^

+ Add filter

Start search

<input type="checkbox"/>	id ⓘ	email	name	registrationDate
<input type="checkbox"/>	001	tblue@noServer.com	Tom Blue	1586217600
<input type="checkbox"/>	091	swhite@noServer.com	Sarah White	1586217600
<input type="checkbox"/>	111	tblue@noServer.com	Tom Blue	1586217600
<input type="checkbox"/>	888	swhite@noServer.com	Sarah White	1586217600

Elimine los recursos de AWS

Enhorabuena, ha creado un flujo de trabajo de AWS sin servidor utilizando SDK para Java de AWS. Como se indicó al principio de este tutorial, asegúrese de cancelar todos los recursos que cree mientras realiza este tutorial para asegurarse de que no se le cobre nada. Para ello, puede eliminar la pila de AWS CloudFormation que creó en el tema [Crear los recursos de AWS](#) de este tutorial, de la siguiente manera:

1. Abra [AWS CloudFormation en la consola de administración de AWS](#).
2. Abra la página Pilas y seleccione la pila.
3. Elija Eliminar.

Crear eventos programados para ejecutar funciones de Lambda

Puedes crear un evento programado que invoque una AWS Lambda función mediante un Amazon CloudWatch Event. Puede configurar un CloudWatch evento para que utilice una expresión cron para programar cuándo se invoca una función Lambda. Por ejemplo, puede programar un CloudWatch evento para que invoque una función Lambda todos los días de la semana.

Lambda es un servicio de computación que permite ejecutar código sin aprovisionar ni administrar servidores. Puede crear funciones de Lambda en varios lenguajes de programación. Para obtener más información acerca de Lambda, consulte [Qué es Lambda](#).

En este tutorial, creará una función de Lambda mediante la API de tiempo de ejecución de JavaScript Lambda. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. Por ejemplo, supongamos que una organización envía un mensaje de texto al móvil a sus empleados para felicitarles por su primer aniversario, como se muestra en esta ilustración.

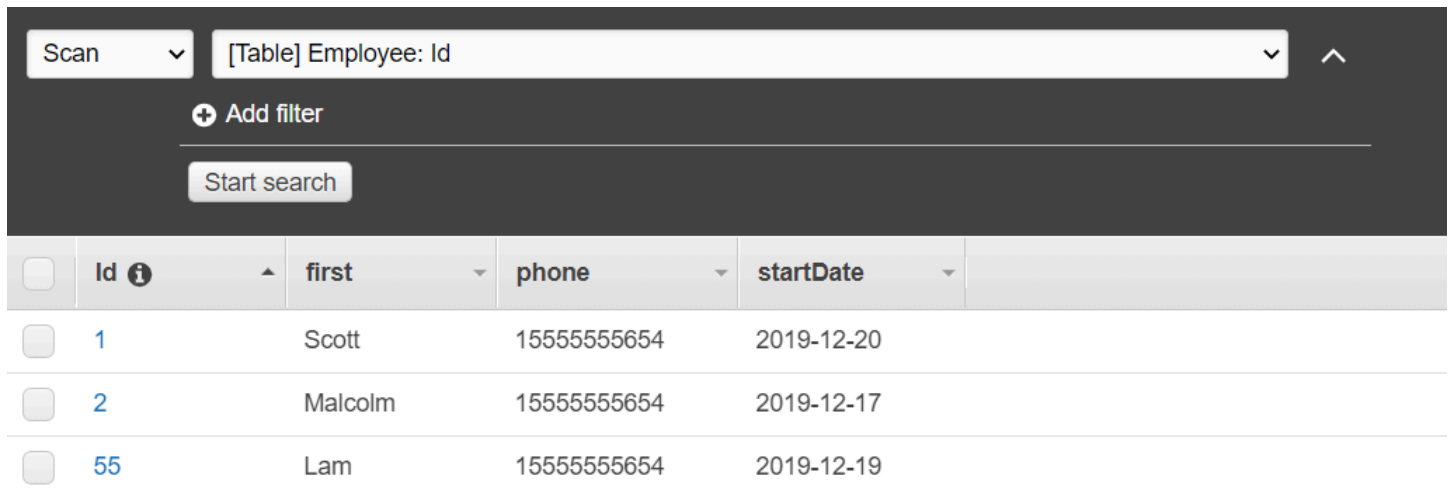


Completar el tutorial debería tomarle aproximadamente 20 minutos.

En este tutorial, se muestra cómo usar la JavaScript lógica para crear una solución que ejecute este caso de uso. Por ejemplo, aprenderá a leer una base de datos para determinar qué empleados han cumplido un año, cómo procesar los datos y cómo enviar un mensaje de texto, todo ello mediante una función de Lambda. A continuación, aprenderá a utilizar una expresión cron para invocar la función de Lambda todos los días de la semana.

En este tutorial de AWS utiliza una tabla de Amazon DynamoDB denominada Employee que contiene estos campos.

- id: la clave principal de la tabla.
- firstName: nombre del empleado.
- teléfono: número de teléfono del empleado.
- startDate: fecha de inicio del empleado.



	Id <i>i</i>	first	phone	startDate
<input type="checkbox"/>	1	Scott	15555555654	2019-12-20
<input type="checkbox"/>	2	Malcolm	15555555654	2019-12-17
<input type="checkbox"/>	55	Lam	15555555654	2019-12-19

Important

Costo de finalización: los servicios de AWS incluidos en este documento están incluidos en la suscripción gratuita de AWS. Sin embargo, asegúrese de cancelar todos los recursos después de haber completado este tutorial para asegurarse de que no se le cobre nada.

Para crear la app:

1. [Completar los requisitos previos](#)
2. [Crear los recursos de AWS](#)

3. [Preparar el script del navegador](#)
4. [Crear y cargar una función de Lambda](#)
5. [Implementar la función de Lambda](#)
6. [Ejecutar la aplicación](#)
7. [Eliminar los recursos](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos TypeScript ejemplos de Node.js e instale los módulos necesarios AWS SDK for JavaScript y de terceros. Siga las instrucciones que figuran en [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia las herramientas y los SDK de AWS.

Crear los recursos de AWS

Este tutorial requiere los siguientes recursos.

- Una tabla de Amazon DynamoDB llamada Empleado con una clave llamada Id y los campos que se muestran en la ilustración anterior. Asegúrese de introducir los datos correctos, incluido un teléfono móvil válido con el que desee probar este caso de uso. Para obtener más información, consulte [Crear una tabla](#).
- Un rol de IAM con permisos adjuntos para ejecutar funciones de Lambda.
- Un bucket de Amazon S3 para alojar la función de Lambda.

Puede crear estos recursos manualmente, pero le recomendamos que los aprovisiones utilizando AWS CloudFormation tal y cómo se describe en este tutorial.

Crear los recursos de AWS usando AWS CloudFormation

AWS CloudFormation le permite crear y aprovisionar implementaciones de infraestructura de AWS de forma predecible y uniforme. Para obtener más información sobre AWS CloudFormation, consulte la [Guía del usuario de AWS CloudFormation](#).

Para crear la pila de AWS CloudFormation usando AWS CLI:

1. Para instalar y configurar AWS CLI, siga las instrucciones de la [Guía del usuario de AWS CLI](#).
2. Cree un archivo con un nombre `setup.yaml` en el directorio raíz de la carpeta de su proyecto y copie [el contenido GitHub en él](#).

Note

La AWS CloudFormation plantilla se generó utilizando lo que AWS CDK está [disponible aquí GitHub](#). Para obtener más información acerca de AWS CDK, consulte la [Guía para desarrolladores de AWS Cloud Development Kit \(CDK\)](#).

3. Ejecute el siguiente comando desde la línea de comandos y reemplace `STACK_NAME` por un nombre único para la pila.

Important

El nombre de la pila tiene que ser único en una región de AWS de una cuenta de AWS. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

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

Para obtener más información sobre los parámetros de los comandos `create-stack`, consulte la [Guía de referencia de comandos de AWS CLI](#) y la [Guía del usuario de AWS CloudFormation](#).

Para ver una lista de los recursos de la consola, abra la pila en el panel de control de AWS CloudFormation y seleccione la pestaña Recursos. Los necesitará para el tutorial.

4. Cuando la pila esté creada, utilice AWS SDK for JavaScript para rellenar la tabla de DynamoDB, tal y como se describe en [Rellenar la tabla de DynamoDB](#).

Rellenar la tabla de DynamoDB

Para rellenar la tabla, primero cree un directorio con el nombre `libs` y, dentro de él, cree un archivo con el nombre `dynamoClient.js` y pegue en él el contenido siguiente.

```
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 };
```

Este código está disponible [aquí en GitHub](#).

A continuación, cree un archivo con un nombre `populate-table.js` en el directorio raíz de la carpeta de su proyecto y copie el contenido [aquí GitHub en](#) él. Para uno de los elementos, sustituya el valor de la propiedad `phone` por un número de teléfono móvil válido con el formato E.164, y el valor de `startDate` por la fecha de hoy.

Desde la línea de comandos, ejecute el comando siguiente:

```
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();

```

Este código está disponible [aquí en GitHub](#).

Creación de la función de Lambda

Configuración del SDK

En primer lugar, importe los módulos y comandos necesarios de AWS SDK for JavaScript (v3): `DynamoDBClient` y `ScanCommand` de `DynamoDB`, y `SNSClient` el comando `PublishCommand` de `Amazon SNS`. Sustituya **REGION** por la región de AWS. A continuación, calcule la fecha de hoy y asígnela a un parámetro. A continuación, cree los parámetros para `ScanCommand`. Sustituya

TABLE_NAME por el nombre de la tabla que creó en la sección [Crear los recursos de AWS](#) de este ejemplo.

El siguiente fragmento de código muestra este paso. (Consulte [Agrupación de la función de Lambda](#) para ver el ejemplo 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",
};
```

Escaneo de la tabla de DynamoDB

Primero cree una función `async/await` llamada `sendText` para publicar un mensaje de texto usando `PublishCommand` de Amazon SNS. A continuación, añada un patrón de bloques `try` que escanee la tabla de DynamoDB en busca de empleados cuyo aniversario laboral sea hoy y, a continuación, llame a la función `sendText` para que envíe un mensaje de texto a estos empleados. Si se produce un error, se llama al bloque `catch`.

El siguiente fragmento de código muestra este paso. (Consulte [Agrupación de la función de Lambda](#) para ver el ejemplo 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);
  }
};
```

Agrupación de la función de Lambda

En este tema se describe cómo agrupar `mylambdafunction.js` y los módulos AWS SDK for JavaScript necesarios para este ejemplo en un archivo de agrupación llamado `index.js`.

1. Si aún no lo ha hecho, siga [Tareas previas necesarias](#) para este ejemplo para instalar Webpack.

Note

Para obtener más información sobre Webpack, consulte [Combine aplicaciones con webpack](#).

2. Ejecute lo siguiente en la línea de comandos para agrupar el JavaScript objeto de este ejemplo en un archivo llamado `index.js`:

```
webpack mylambdafunction.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

Important

Observe que la salida tiene el nombre `index.js`. Esto se debe a que las funciones de Lambda tienen que tener un controlador `index.js` para funcionar.

3. Comprima el archivo de salida empaquetado, `index.js`, en un archivo ZIP denominado `my-lambda-function.zip`.
4. Suba `mylambdafunction.zip` al bucket de Amazon S3 que creó en el tema [Crear los recursos de AWS](#) de este tutorial.

Este es el código completo del script del 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);
  }
}
```

```
}  
};
```

Implementar la función de Lambda

En la raíz del proyecto, cree un archivo `lambda-function-setup.js` y pegue en él el contenido siguiente.

Sustituya `BUCKET_NAME` por el nombre del bucket de Amazon S3 en el que cargó la versión ZIP de la función de Lambda. Sustituya `ZIP_FILE_NAME` por el nombre de la versión ZIP de la función de Lambda. Sustituya `IAM_ROLE_ARN` por el número de recurso de Amazon (ARN) del rol de IAM que creó en el tema [Crear los recursos de AWS](#) de este tutorial. Sustituya `LAMBDA_FUNCTION_NAME` por un nombre para la función de 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();
```

Introduzca lo siguiente en la línea de comandos para implementar la función de Lambda.

```
node lambda-function-setup.js
```

Este ejemplo de código está disponible [aquí en GitHub](#).

Configurar CloudWatch para invocar las funciones Lambda

CloudWatch Para configurar la invocación de las funciones Lambda:

1. Abra la página de Funciones en la consola de Lambda.
2. Elija la función Lambda.
3. En Diseñador, elija Añadir disparador.
4. Establezca el tipo de disparador en Events/CloudWatch . EventBridge
5. En Regla, seleccione Crear una nueva regla.
6. Rellene el nombre y la descripción de la regla.
7. Para el tipo de regla, seleccione Expresión de programación.
8. En el campo Expresión de programación, introduzca una expresión cron. Por ejemplo, cron(0 12 ? * MON-FRI *).
9. Elija Agregar.

Note

Para obtener más información, consulte [Uso de Lambda con CloudWatch eventos](#).

Elimine los recursos

¡Enhorabuena! Ha invocado una función Lambda a través de eventos CloudWatch programados de Amazon mediante AWS SDK for JavaScript. Como se indicó al principio de este tutorial, asegúrese de cancelar todos los recursos que cree mientras realiza este tutorial para asegurarse de que no se le cobre nada. Para ello, puede eliminar la pila de AWS CloudFormation que creó en el tema [Crear los recursos de AWS](#) de este tutorial, de la siguiente manera:

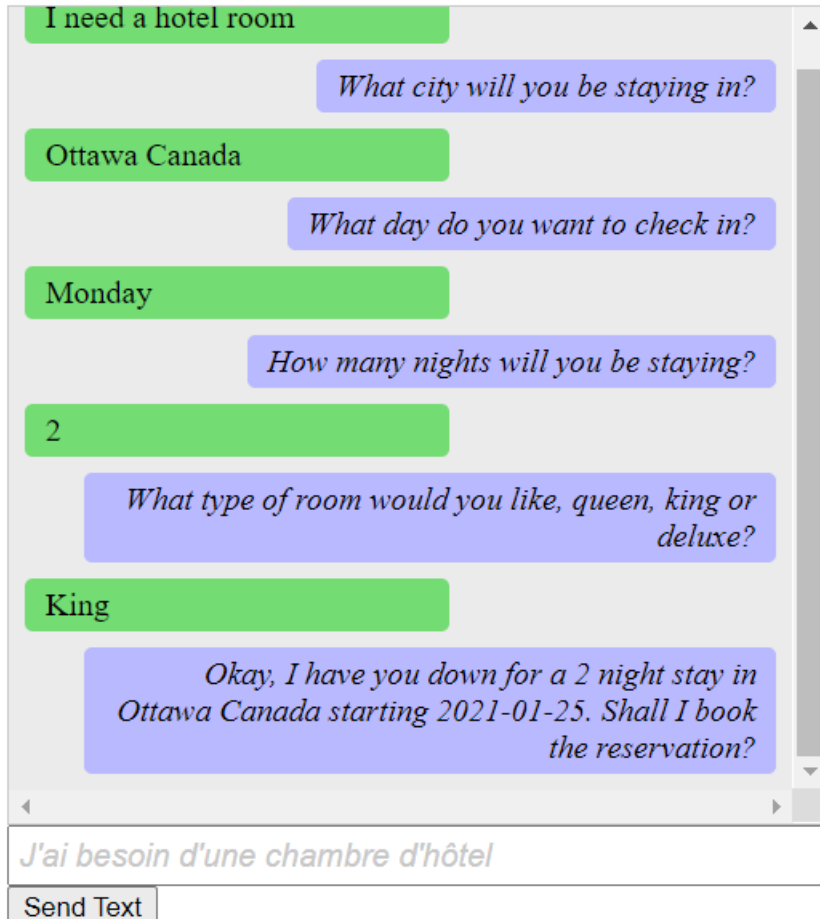
1. Abra la [consola de AWS CloudFormation](#).
2. En la página Pilas, seleccione la pila.
3. Elija Eliminar.

Creación de un chatbot de Amazon Lex

Puede crear un chatbot de Amazon Lex dentro de una aplicación web para atraer a los visitantes de su sitio web. Un chatbot de Amazon Lex es una funcionalidad que realiza conversaciones de chat en línea con los usuarios sin proporcionar contacto directo con una persona. Por ejemplo, en la siguiente ilustración se muestra un chatbot de Amazon Lex que interactúa con un usuario para que reserve una habitación de hotel.

Amazon Lex - BookTrip

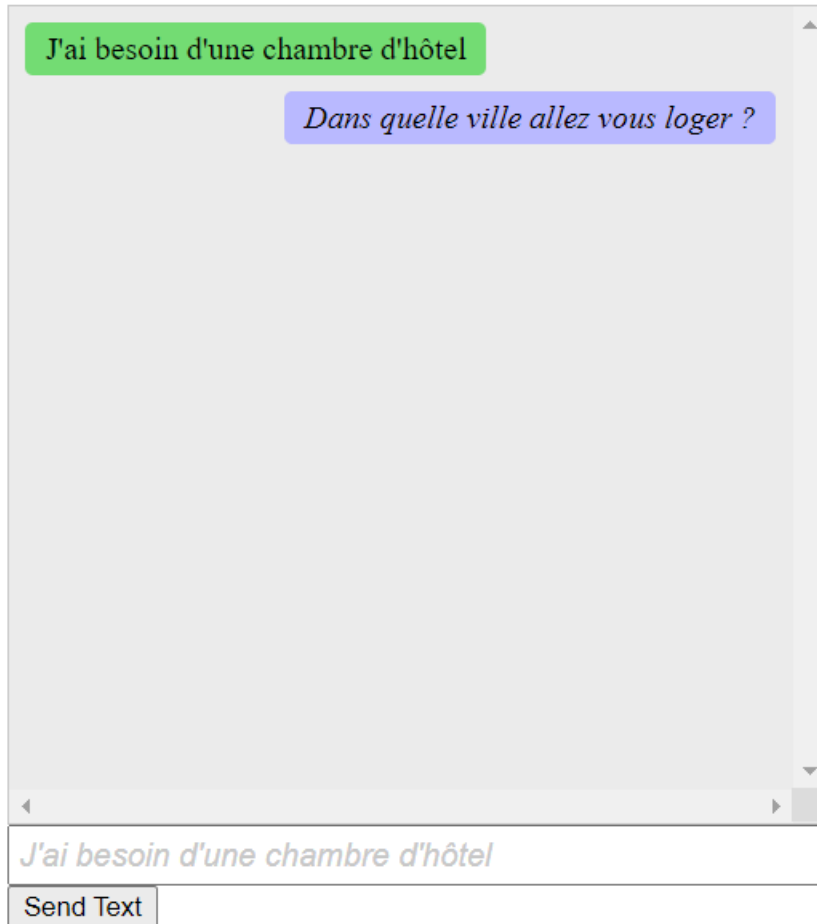
This multiple language chatbot shows you how easy it is to incorporate [Amazon Lex](#) into your web apps. Try it out.



El chatbot de Amazon Lex creado en este tutorial de AWS es capaz de manejar varios idiomas. Por ejemplo, un usuario que habla francés puede introducir texto en francés y obtener una respuesta en francés.

Amazon Lex - BookTrip

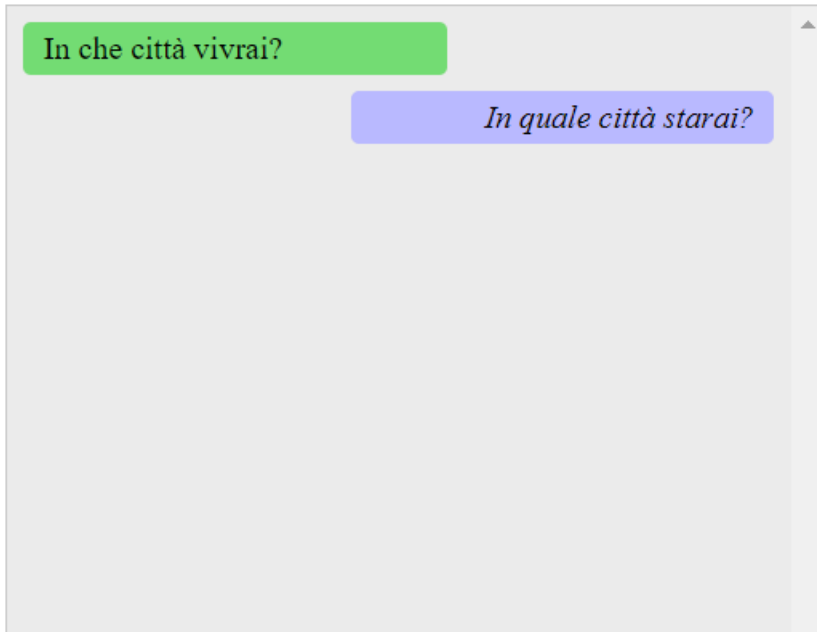
This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



Del mismo modo, un usuario puede comunicarse con el chatbot de Amazon Lex en italiano.

Amazon Lex - BookTrip

This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



Este tutorial de AWS le guía a través de la creación de un chatbot de Amazon Lex y su integración en una aplicación web de Node.js. La AWS SDK for JavaScript (v3) se usa para invocar estos AWS servicios:

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

Costo de la prestación: los servicios de AWS incluidos en este documento están incluidos en la [suscripción gratuita de AWS](#).

Nota: Recuerde cancelar todos los recursos que cree mientras sigue este tutorial para asegurarse de que no se le cobre nada.

Para crear la aplicación:

1. [Requisitos previos](#)
2. [Aprovisionar recursos](#)

3. [Cree el chatbot de Amazon Lex](#)
4. [Cree el HTML](#)
5. [Cree el script del navegador](#)
6. [Pasos siguientes](#)

Requisitos previos

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos TypeScript ejemplos de nodos e instale los módulos necesarios AWS SDK for JavaScript y de terceros. Siga las instrucciones de [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de las herramientas y los SDK de AWS.

Important

Este ejemplo usa ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

No obstante, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Crear los recursos de AWS

Este tutorial requiere los siguientes recursos.

- Un rol de IAM no autenticado con permisos adjuntos para:
 - Amazon Comprehend
 - Amazon Translate
 - Amazon Lex


Puede crear estos recursos manualmente, pero le recomendamos que los aprovisiones usando AWS CloudFormation tal y como se describe en este tutorial.

Crear los recursos de AWS usando AWS CloudFormation

AWS CloudFormation le permite crear y aprovisionar implementaciones de infraestructura de AWS de forma predecible y uniforme. Para obtener más información sobre AWS CloudFormation, consulte la [Guía del usuario de AWS CloudFormation](#).


Para crear la pila de AWS CloudFormation usando AWS CLI:

1. Para instalar y configurar AWS CLI, siga las instrucciones de la [Guía del usuario de AWS CLI](#).
2. Cree un archivo con un nombre `setup.yaml` en el directorio raíz de la carpeta de su proyecto y copie [el contenido GitHub en él](#).

 Note

La AWS CloudFormation plantilla se generó utilizando lo que AWS CDK está [disponible aquí GitHub](#). Para obtener más información acerca de AWS CDK, consulte la [Guía para desarrolladores de AWS Cloud Development Kit \(CDK\)](#).

3. Ejecute el siguiente comando desde la línea de comandos y reemplace `STACK_NAME` por un nombre único para la pila.

 Important

El nombre de la pila tiene que ser único en una región de AWS de una cuenta de AWS. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

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

Para obtener más información sobre los parámetros de los comandos `create-stack`, consulte la [Guía de referencia de comandos AWS CLI](#) y la [Guía del usuario de AWS CloudFormation](#).

Para ver los recursos creados, abra la consola de Amazon Lex, elija la pila y seleccione la pestaña Recursos.

Crear un bot de Amazon Lex

⚠ Important

Utilice la versión 1 de la consola de Amazon Lex para crear el bot. Este ejemplo no funciona con los bots creados con la V2.

El primer paso es crear un chatbot de Amazon Lex mediante la consola de administración de Amazon Web Services. En este ejemplo, se utiliza el BookTripejemplo de Amazon Lex. Para obtener más información, consulte [Book Trip](#).

- Inicie sesión en la consola de administración de Amazon Web Services y abra la consola de Amazon Lex en la [consola de Amazon Web Services](#).
- En la página Bots, elija Crear.
- Elija el BookTripejemplo (deje el nombre del bot predeterminado BookTrip).

Create your bot

Amazon Lex enables any developer to build conversational chatbots quickly and easily. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversational flow directly from the console, and then Amazon Lex manages the dialogue and dynamically adjusts the response. To get started, you can choose one of the sample bots provided below or build a new custom bot from scratch.

The screenshot shows the 'Create your bot' interface. Under the 'CREATE YOUR OWN' section, there are four buttons: 'Custom bot', 'BookTrip', 'OrderFlowers', and 'ScheduleAppointment'. The 'BookTrip' button is highlighted with a red border. Below these buttons, there is a 'Bot name' input field, which also contains 'BookTrip' and is highlighted with a red border.



- Rellena la configuración predeterminada y selecciona Crear (la consola muestra el BookTripebot). En la pestaña Editor, revise los detalles de la intención preconfigurada ().
- Pruebe el bot en la ventana de pruebas. Comience la prueba escribiendo Quiero reservar una habitación de hotel.

[> Test bot \(Latest\)](#)

✔ Ready. Build complete

I want to book a hotel room

What city will you be staying in?

[Clear chat history](#)

 *Chat with your bot...*

Inspect response

Dialog State: ElicitSlot

[Hid](#)

Summary Detail

Intent: BookHotel

- Seleccione Publicar y especifique un nombre de alias (necesitará este valor cuando utilice AWS SDK for JavaScript).

Note

Debes hacer referencia al nombre y al alias del bot en tu JavaScript código.

Cree el HTML

Cree un archivo denominado `index.html`. Copie y pegue el siguiente código en `index.html`. Este HTML hace referencia a `main.js`. Se trata de una versión agrupada de `index.js`, que incluye los módulos de AWS SDK for JavaScript necesarios. Creará este archivo en [Cree el HTML](#). `index.html` también hace referencia a `style.css`, que añade los 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>
```


Este código también está disponible [aquí en GitHub](#).

Cree el script del navegador

Cree un archivo denominado `index.js`. Copie y pegue el siguiente código `index.js` Importe los módulos y comandos de AWS SDK for JavaScript necesarios. Cree clientes para Amazon Lex, Amazon Comprehend y Amazon Translate. Reemplace **REGION** por la región de AWS y **IDENTITY_POOL_ID** por el ID del grupo de identidades que creó en [Crear los recursos de AWS](#). Para recuperar este ID del grupo de identidades, abra el grupo de identidades en la consola de Amazon Cognito, seleccione Editar grupo de identidades y elija Código de muestra en el menú lateral. Anote el ID del grupo de identidades que se muestra en rojo en la consola.

En primer lugar, cree un directorio `libs` y cree los objetos de cliente de servicio necesarios mediante la creación de tres archivos, `comprehendClient.js`, `lexClient.js` y `translateClient.js`.

Pegue el código correspondiente que aparece a continuación en cada uno de ellos y sustituya **REGION** e **IDENTITY_POOL_ID** en cada archivo.

 Note

Use el ID del grupo de identidades de Amazon Cognito que creó en [Crear los recursos de AWS usando 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 };
```

Este código está disponible [aquí en GitHub](#).

A continuación, cree un archivo `index.js` y pegue en él el siguiente código.

Sustituya ***BOT_ALIAS*** y ***BOT_NAME*** por el alias y el nombre de su bot de Amazon Lex, respectivamente, y ***USER_ID*** por un ID de usuario. La función asíncrona de `createResponse` hace lo siguiente:

- Toma el texto introducido por el usuario en el navegador y utiliza Amazon Comprehend para determinar el código de su idioma.
- Toma el código del idioma y utiliza Amazon Translate para traducir el texto al inglés.
- Toma el texto traducido y utiliza Amazon Lex para generar una respuesta.
- Publica la respuesta en la página del 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;
```

Este código está disponible [aquí en GitHub](#).

Ahora use Webpack para agrupar los módulos `index.js` y de AWS SDK for JavaScript en un solo archivo, `main.js`.

1. Si aún no lo ha hecho, siga [Requisitos previos](#) para este ejemplo para instalar Webpack.

Note

Para obtener más información sobre Webpack, consulte [Combine aplicaciones con webpack](#).

2. Ejecute lo siguiente en la línea de comandos JavaScript para agrupar este ejemplo en un archivo llamado `main.js`:

```
webpack index.js --mode development --target web --devtool false -o main.js
```

Siguientes pasos

¡Enhorabuena! Ha creado una aplicación Node.js que utiliza Amazon Lex para crear una experiencia de usuario interactiva. Como se indicó al principio de este tutorial, asegúrese de cancelar todos los

recursos que cree mientras realiza este tutorial para asegurarse de que no se le cobre nada. Para ello, puede eliminar la pila de AWS CloudFormation que creó en el tema [Crear los recursos de AWS](#) de este tutorial, de la siguiente manera:

1. Abra la [consola de AWS CloudFormation](#).
2. En la página Pilas, seleccione la pila.
3. Elija Eliminar.

Para ver más ejemplos de servicios cruzados de AWS, consulte [ejemplos de servicios cruzados de AWS SDK for JavaScript](#).

Creación de una aplicación de mensajería de ejemplo

Puede crear una aplicación de AWS que envíe y recupere mensajes usando AWS SDK for JavaScript y Amazon Simple Queue Service (Amazon SQS). Los mensajes se almacenan en una cola del tipo “primero en entrar, primero en salir” (FIFO), lo que garantiza que el orden de los mensajes sea coherente. Por ejemplo, el primer mensaje que se almacena en la cola es el primer mensaje que se lee de la cola.

Note

Para obtener más información sobre Amazon SQS, consulte [¿Qué es Amazon Simple Queue Service?](#)

En este tutorial, deberá crear una aplicación de Node.js llamada Mensajería de AWS.

Costo de finalización: los servicios de AWS incluidos en este documento están incluidos en la [suscripción gratuita de AWS](#).

Nota: Recuerde cancelar todos los recursos que cree mientras sigue este tutorial para asegurarse de que no se le cobre nada.

Para crear la aplicación:

1. [Requisitos previos](#)
2. [Aprovisionar recursos](#)
3. [Comprenda el flujo de trabajo](#)

4. [Cree el HTML](#)
5. [Cree el script del navegador](#)
6. [Pasos siguientes](#)

Requisitos previos

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Configure el entorno del proyecto para ejecutar estos TypeScript ejemplos de nodos e instale los módulos necesarios AWS SDK for JavaScript y de terceros. Siga las instrucciones de [GitHub](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información sobre cómo proporcionar un archivo de credenciales compartido, consulte [Archivos de configuración y credenciales compartidos](#) en la Guía de referencia de las herramientas y los SDK de AWS.

Important

Este ejemplo usa ECMAScript6 (ES6). Esto requiere la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#).

No obstante, si prefiere utilizar la sintaxis CommonJS, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Crear los recursos de AWS

Este tutorial requiere los siguientes recursos:

- Un rol de IAM no autenticado con permisos para Amazon SQS.
- Una cola FIFO de Amazon SQS llamada Message.fifo: para obtener información sobre la creación de una cola, consulte [Crear una cola de Amazon SQS](#).

Puede crear estos recursos manualmente, pero le recomendamos que los aprovisiones usando AWS CloudFormation (AWS CloudFormation), tal y como se describe en este tutorial.

 Note


AWS CloudFormation es un marco de desarrollo de software que le permite definir los recursos de las aplicaciones en la nube. Para obtener más información, consulte la [Guía del usuario de AWS CloudFormation](#).

Crear los recursos de AWS usando AWS CloudFormation

AWS CloudFormation le permite crear y aprovisionar implementaciones de infraestructura de AWS de forma predecible y uniforme. Para obtener más información sobre AWS CloudFormation, consulte la [Guía del usuario de AWS CloudFormation](#).

Para crear la pila de AWS CloudFormation usando AWS CLI:

1. Para instalar y configurar AWS CLI, siga las instrucciones de la [Guía del usuario de AWS CLI](#).
2. Cree un archivo con un nombre `setup.yaml` en el directorio raíz de la carpeta de su proyecto y copie [el contenido GitHub en él](#).

 Note

La AWS CloudFormation plantilla se generó utilizando lo que AWS CDK está [disponible aquí GitHub](#). Para obtener más información acerca de AWS CDK, consulte la [Guía para desarrolladores de AWS Cloud Development Kit \(CDK\)](#).

3. Ejecute el siguiente comando desde la línea de comandos y reemplace `STACK_NAME` por un nombre único para la pila.

 Important

El nombre de la pila tiene que ser único en una región de AWS de una cuenta de AWS. El nombre puede tener una longitud de hasta 128 caracteres, y se permiten números y guiones.

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

Para obtener más información sobre los parámetros de los comandos `create-stack`, consulte la [Guía referencia de comandos de AWS CLI](#) y la [Guía del usuario de AWS CloudFormation](#).

Para ver los recursos creados, abra AWS CloudFormation en la consola de administración de AWS, elija la pila y seleccione la pestaña Recursos.

Comprender la aplicación de mensajería de AWS

Para enviar un mensaje a una cola de SQS, introdúzcalo en la aplicación y seleccione Enviar.

Una vez enviado el mensaje, la aplicación lo mostrará.

Puede elegir Depuración para purgar los mensajes de la cola de Amazon SQS. Esto dará como resultado una cola vacía y no se mostrará ningún mensaje en la aplicación.

A continuación se describe cómo gestiona la aplicación un mensaje:

- El usuario selecciona su nombre, introduce su mensaje y envía el mensaje, lo que inicia la función `pushMessage`.
- `pushMessage` recupera la URL de la cola de Amazon SQS y, a continuación, envía un mensaje con un valor de ID de mensaje único (un GUID), el texto del mensaje y el usuario a la cola de Amazon SQS.
- `pushMessage` recupera los mensajes de la cola de Amazon SQS, extrae el usuario y el mensaje de cada mensaje y muestra los mensajes.
- El usuario puede purgar los mensajes, lo que los elimina de la cola de Amazon SQS y de la interfaz de usuario.

Crear la página HTML

Ahora puede crear los archivos HTML necesarios para la interfaz gráfica de usuario (GUI) de la aplicación. Cree un archivo denominado `index.html`. Copie y pegue el siguiente código en `index.html`. Este HTML hace referencia a `main.js`. Se trata de una versión agrupada de `index.js`, que incluye los módulos de AWS SDK for JavaScript necesarios.

```
<!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>
```

```
</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>
  </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">
    
    <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>
```

Este código también está disponible [aquí en GitHub](#).

Crear el script del navegador

En este tema, creará un script de navegador para la aplicación. Cuando haya creado el script del navegador, lo agrupará en un archivo llamado `main.js` como se describe en [Agrupar el JavaScript](#).

Cree un archivo denominado `index.js`. Copia y pega el código de [aquí en GitHub](#) adelante en él.

Este código se explica en las siguientes secciones:

1. [Configuración](#)
2. [populateChat](#)
3. [pushmessages](#)
4. [purgar](#)

Configuración

En primer lugar, cree un directorio `libs` y cree el objeto de cliente Amazon SQS necesario mediante la creación de un archivo llamado `sqsClient.js`. Sustituya *REGION* e *IDENTITY_POOL_ID* en cada uno de ellos.

Note

Utilice el ID del grupo de identidades de Amazon Cognito que creó en [Crear los recursos de 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
  }),
});
```

En `index.js`, importe los módulos y comandos de AWS SDK for JavaScript necesarios. Reemplace `SQS_QUEUE_NAME` por el nombre de la cola de Amazon SQS que creó en [Crear los recursos de 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

La función `onload populateChat` recupera automáticamente la URL de la cola de Amazon SQS, y recupera todos los mensajes de la cola y los muestra.

```
$(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 =
      '';
    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);
}
};

```

Mensajes push

El usuario selecciona su nombre, introduce su mensaje y envía el mensaje, lo que inicia la función `pushMessage`. `pushMessage` recupera la URL de la cola de Amazon SQS y, a continuación, envía un mensaje con un valor de ID de mensaje único (un GUID), el texto del mensaje y el usuario a la cola de Amazon SQS. A continuación, recupera todos los mensajes de la cola de Amazon SQS y los muestra.

```

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 =
    '';
  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;

```

Purgar mensajes

purge elimina los mensajes de la cola de Amazon SQS y de la interfaz de usuario.

```

// 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;
```

Agrupar el JavaScript

[Este código de script de navegador completo está disponible aquí en GitHub](#) .

Ahora use Webpack para agrupar los módulos `index.js` y de AWS SDK for JavaScript en un solo archivo, `main.js`.

1. Si aún no lo ha hecho, siga [Requisitos previos](#) para este ejemplo para instalar Webpack.

Note

Para obtener más información sobre Webpack, consulte [Combine aplicaciones con webpack](#).

2. Ejecute lo siguiente en la línea de comandos JavaScript para agrupar este ejemplo en un archivo llamado `<index.js>`:

```
webpack index.js --mode development --target web --devtool false -o main.js
```

Siguientes pasos

¡Enhorabuena! Ha creado e implementado la aplicación de mensajería de AWS que utiliza Amazon SQS. Como se indicó al principio de este tutorial, recuerde cancelar todos los recursos que cree mientras realiza este tutorial para asegurarse de que no se le cobre nada. Para ello, puede eliminar

la pila de AWS CloudFormation que creó en el tema [Crear los recursos de AWS](#) de este tutorial, de la siguiente manera:

1. Abra [AWS CloudFormation en la consola de administración de AWS](#).
2. Abra la página Pilas y seleccione la pila.
3. Elija Eliminar.

AWS Cloud9 Utilízalo con AWS SDK for JavaScript

Puede usarlo AWS Cloud9 AWS SDK for JavaScript para escribir y ejecutar el código del navegador, así como escribir, ejecutar y depurar el código Node.js, utilizando solo un navegador. JavaScript AWS Cloud9 incluye herramientas como un editor de código y un terminal, además de un depurador para el código Node.js.

Como el AWS Cloud9 IDE está basado en la nube, puede trabajar en sus proyectos desde su oficina, casa o cualquier lugar mediante una máquina conectada a Internet. Para obtener información general al respecto AWS Cloud9, consulte la Guía del [AWS Cloud9 usuario](#).

En los siguientes pasos se describe cómo realizar la configuración AWS Cloud9 con el SDK para JavaScript.

Contenido

- [Paso 1: Configura tu AWS cuenta para usar AWS Cloud9](#)
- [Paso 2: Configura tu entorno de AWS Cloud9 desarrollo](#)
- [Paso 3: Configura el SDK para JavaScript](#)
 - [Para configurar el SDK JavaScript de Node.js](#)
 - [Para configurar el SDK JavaScript en el navegador](#)
- [Paso 4: Descargue el código de ejemplo](#)
- [Paso 5: Ejecute y depure el código de ejemplo](#)

Paso 1: Configura tu AWS cuenta para usar AWS Cloud9

Comience a utilizarla AWS Cloud9 iniciando sesión en la AWS Cloud9 consola como una entidad AWS Identity and Access Management (de IAM) (por ejemplo, un usuario de IAM) que tenga permisos de acceso a su AWS cuenta. AWS Cloud9

Para configurar una entidad de IAM en tu AWS cuenta para acceder AWS Cloud9 a la AWS Cloud9 consola e iniciar sesión en ella, consulta la sección [Configuración del equipo AWS Cloud9 en la Guía del AWS Cloud9 usuario](#).

Paso 2: Configura tu entorno de AWS Cloud9 desarrollo

Después de iniciar sesión en la AWS Cloud9 consola, utilícela para crear un entorno de AWS Cloud9 desarrollo. Tras crear el entorno, AWS Cloud9 abre el IDE de ese entorno.

Consulte [Creación de un entorno en AWS Cloud9](#) en la Guía del usuario de AWS Cloud9 para obtener más información.

Note

Al crear el entorno en la consola por primera vez, le recomendamos que elija la opción `Create a new instance for environment (EC2)` [Crear una nueva instancia para el entorno (EC2)]. Esta opción indica AWS Cloud9 que hay que crear un entorno, lanzar una instancia de Amazon EC2 y, a continuación, conectar la nueva instancia al nuevo entorno. Esta es la forma más rápida de empezar a AWS Cloud9 utilizarla.

Paso 3: Configura el SDK para JavaScript

Después de AWS Cloud9 abrir el IDE de su entorno de desarrollo, siga uno o ambos de los siguientes procedimientos para usar el IDE y configurar el SDK JavaScript en su entorno.

Para configurar el SDK JavaScript de Node.js

1. Si el terminal todavía no está abierto en el IDE, ábralo. Para ello, en la barra de menú del IDE, elija `Ventana, Nuevo terminal`.
2. Ejecute el siguiente comando `npm` para instalarlo en el Cloud9 cliente del SDK para JavaScript.

```
npm install @aws-sdk/client-cloud9
```

Si el IDE no puede encontrar `npm`, ejecute los siguientes comandos de uno en uno, en el orden que se indica a continuación para instalar `npm`. (En estos comandos se presupone que ha elegido la opción `Create a new instance for environment (EC2)` [Crear una nueva instancia para el entorno (EC2)], vista anteriormente en este tema).

⚠ Warning

AWS no controla el siguiente código. Antes de ejecutarlo, asegúrese de comprobar su autenticidad e integridad. Puede encontrar más información sobre este código en el GitHub repositorio [nvm](#) (Node Version Manager).

```
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 el SDK JavaScript en el navegador

Para usar el SDK JavaScript en sus páginas HTML, úselo para WebPack agrupar los módulos de cliente necesarios y todas JavaScript las funciones necesarias en un solo JavaScript archivo, y agréguelo en una etiqueta <head> de script en las páginas HTML. Por ejemplo:

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

i Note

Para obtener más información sobre Webpack, consulte [Combine aplicaciones con webpack](#)

Paso 4: Descargue el código de ejemplo

Usa la terminal que abriste en el paso anterior para descargar el código de ejemplo del SDK JavaScript al entorno de AWS Cloud9 desarrollo. (Si el terminal todavía no está abierto en el IDE, ábralo eligiendo Window, New Terminal (Ventana, nuevo terminal) en la barra de menús del IDE).

Ejecute el comando siguiente para descargar el código de ejemplo. Este comando descarga una copia de todos los ejemplos de código utilizados en la documentación oficial del AWS SDK en el directorio raíz de su entorno.


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

Para encontrar ejemplos de código para el SDK JavaScript, usa la ventana Entorno para abrir el `ENVIRONMENT_NAME\aws-doc-sdk-examples\javascriptv3\example_code/src`, donde *ENVIRONMENT_NAME es el nombre* de tu entorno de AWS Cloud9 desarrollo.

Para aprender a trabajar con estos y otros ejemplos de código, consulta el [SDK para ver ejemplos de JavaScript código](#).

Paso 5: Ejecute y depure el código de ejemplo

Para ejecutar código en su entorno de AWS Cloud9 desarrollo, consulte [Ejecute su código](#) en la Guía del AWS Cloud9 usuario.

Para depurar código de Node.js, consulte [Depurar su código](#) en la Guía del usuario de AWS Cloud9.

Ejemplos de código de SDK para JavaScript (v3)

Los ejemplos de código de este tema muestran cómo usar la AWS SDK for JavaScript (v3) con. AWS

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Los ejemplos entre servicios son aplicaciones de muestra que funcionan en varios Servicios de AWS.

Ejemplos

- [Acciones y escenarios que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de servicios cruzados que utilizan el SDK para JavaScript \(v3\)](#)

Acciones y escenarios que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con. Servicios de AWS

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Servicios

- [Ejemplos de Auto Scaling con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon Bedrock que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon Bedrock Runtime con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Agents for Amazon Bedrock que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Agents for Amazon Bedrock Runtime que utilizan el SDK para JavaScript \(v3\)](#)
- [CloudWatch ejemplos de uso del SDK para JavaScript \(v3\)](#)

- [CloudWatch Ejemplos de eventos que utilizan el SDK para JavaScript \(v3\)](#)
- [CloudWatch Registra ejemplos con el SDK para JavaScript \(v3\)](#)
- [CodeBuild ejemplos de uso del SDK para JavaScript \(v3\)](#)
- [Ejemplos de proveedores de identidad de Amazon Cognito que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon DocumentDB que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de DynamoDB que utilizan el SDK JavaScript para \(v3\)](#)
- [Ejemplos de Amazon EC2 con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Elastic Load Balancing: versión 2 con el SDK para JavaScript \(v3\)](#)
- [EventBridge ejemplos de uso del SDK para JavaScript \(v3\)](#)
- [AWS Glue ejemplos de uso del SDK para JavaScript \(v3\)](#)
- [HealthImaging ejemplos que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de IAM que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Kinesis con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Lambda con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon Personalize con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon Personalize Events con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon Personalize Runtime mediante el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon Pinpoint con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon Redshift con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon S3 que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de S3 Glacier que utilizan el SDK para JavaScript \(v3\)](#)
- [SageMaker ejemplos de uso del SDK para JavaScript \(v3\)](#)
- [Ejemplos de Secrets Manager con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon SES que utilizan el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon SNS con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Amazon SQS con el SDK para JavaScript \(v3\)](#)
- [Ejemplos de Step Functions con el SDK para JavaScript \(v3\)](#)
- [AWS STS ejemplos de uso del SDK para JavaScript \(v3\)](#)
- [AWS Support ejemplos de uso del SDK para JavaScript \(v3\)](#)

- [Ejemplos de Amazon Transcribe con el SDK para JavaScript \(v3\)](#)

Ejemplos de Auto Scaling con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Auto Scaling.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

AttachLoadBalancerTargetGroups

En el siguiente ejemplo de código, se muestra cómo usar `AttachLoadBalancerTargetGroups`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const client = new AutoScalingClient({});
await client.send(
  new AttachLoadBalancerTargetGroupsCommand({
```

```
    AutoScalingGroupName: NAMES.autoScalingGroupName,  
    TargetGroupARNs: [state.targetGroupArn],  
  }},  
);
```

- Para obtener más información sobre la API, consulta la [AttachLoadBalancerTargetsección Grupos](#) en la referencia de la AWS SDK for JavaScript API.

Escenarios

Creación y administración de un servicio resiliente

El siguiente ejemplo de código muestra cómo crear un servicio web con equilibrio de carga que muestre recomendaciones de libros, películas y canciones. El ejemplo muestra cómo responde el servicio a los errores y cómo reestructurarlo para aumentar la resiliencia cuando se produzcan errores.

- Utilice un grupo de Amazon EC2 Auto Scaling para crear instancias de Amazon Elastic Compute Cloud (Amazon EC2) basadas en una plantilla de lanzamiento y para mantener el número de instancias dentro de un rango específico.
- Administre y distribuya las solicitudes HTTP con Elastic Load Balancing.
- Supervise el estado de las instancias de un grupo de escalado automático y reenvíe las solicitudes solo a las instancias en buen estado.
- Ejecute un servidor web Python en cada instancia de EC2 para administrar las solicitudes HTTP. El servidor web responde con recomendaciones y comprobaciones de estado.
- Simule un servicio de recomendaciones con una tabla de Amazon DynamoDB.
- Controle la respuesta del servidor web a las solicitudes y las comprobaciones de estado actualizando AWS Systems Manager los parámetros.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecute el escenario interactivo en un símbolo del sistema.

```
#!/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);
}
```

Cree los pasos para implementar todos los 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,
    );
  }
}),
];
```

Cree los pasos para ejecutar la demostración.

```
// 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;
}

```

Cree los pasos para destruir todos los 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 obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for JavaScript .
 - [AttachLoadBalancerTargetGrupos](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstancePerfil](#)
 - [CreateLaunchPlantilla](#)
 - [CreateListener](#)
 - [CreateLoadEquilibrador](#)
 - [CreateTargetGrupo](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstancePerfil](#)
 - [DeleteLaunchPlantilla](#)
 - [DeleteLoadEquilibrador](#)
 - [DeleteTargetGrupo](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZonas](#)
 - [DescribeIamInstanceProfileAsociaciones](#)
 - [DescribeInstances](#)
 - [DescribeLoadEquilibradores](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGrupos](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)
 - [ReplacelamInstanceProfileAsociación](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

Ejemplos de Amazon Bedrock que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Bedrock.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon Bedrock

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon Bedrock.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [ListFoundationlos modelos](#) en la referencia AWS SDK for JavaScript de la API.

Temas

- [Acciones](#)

Acciones

GetFoundationModel

En el siguiente ejemplo de código, se muestra cómo usar `GetFoundationModel`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga detalles sobre un modelo fundacional.

```
// 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 obtener más información sobre la API, consulta el [GetFoundationModelo](#) en la referencia AWS SDK for JavaScript de la API.

ListFoundationModels

En el siguiente ejemplo de código, se muestra cómo usar `ListFoundationModels`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los modelos fundacionales disponibles.

```
// 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 obtener más información sobre la API, consulta [ListFoundationlos modelos](#) en la referencia AWS SDK for JavaScript de la API.

Ejemplos de Amazon Bedrock Runtime con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Bedrock Runtime.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon Bedrock

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon Bedrock.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [InvokeModel](#) la Referencia AWS SDK for JavaScript de la API.

Temas

- [AI21 Labs Jurassic-2](#)
- [Texto de Amazon Titan](#)
- [Anthropic Claude](#)
- [Cohere Command](#)
- [Meta Llama](#)
- [API de Mistral](#)
- [Escenarios](#)

AI21 Labs Jurassic-2

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a AI21 Labs Jurassic-2 mediante la API Converse de Bedrock.

JavaScript SDK para (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a AI21 Labs Jurassic-2 mediante la API Converse de 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 obtener más información sobre la API, consulta Converse en la referencia de la API.AWS SDK for JavaScript](#)

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a AI21 Labs Jurassic-2 mediante la API Invoke Model.

SDK para (v3) JavaScript

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje 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 obtener más información sobre la API, consulte [InvokeModel](#) la referencia de AWS SDK for JavaScript la API.

Texto de Amazon Titan

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Amazon Titan Text mediante la API Converse de Bedrock.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Amazon Titan Text mediante la API Converse de 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 obtener más información sobre la API, consulte [Converse](#) en AWS SDK for JavaScript la referencia de la API.

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Amazon Titan Text mediante la API Converse de Bedrock y cómo procesar el flujo de respuestas en tiempo real.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Amazon Titan Text mediante la API Converse de Bedrock y procese el flujo de respuestas en tiempo 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 obtener más información sobre la API, consulte la Referencia de [ConverseStream](#) la AWS SDK for JavaScript API.

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Amazon Titan Text mediante la API Invoke Model.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje 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 obtener más información sobre la API, consulte [InvokeModel](#) la referencia de AWS SDK for JavaScript la API.

Anthropic Claude

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Anthropic Claude mediante la API Converse de Bedrock.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Anthropic Claude utilizando la API Converse de 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 obtener más información sobre la API, consulte [Converse](#) en la referencia de la API AWS SDK for JavaScript

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Anthropic Claude mediante la API Converse de Bedrock y procesar el flujo de respuestas en tiempo real.

SDK para (v3 JavaScript)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Anthropic Claude mediante la API Converse de Bedrock y procesa el flujo de respuestas en tiempo 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 obtener más información sobre la API, consulte la referencia de [ConverseStream](#) la AWS SDK for JavaScript API.

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Anthropic Claude mediante la API Invoke Model.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje 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 obtener más información sobre la API, consulte [InvokeModel](#) la referencia de AWS SDK for JavaScript la API.

InvokeModelWithResponseTransmitir

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a los modelos Anthropic Claude mediante la API Invoke Model e imprimir el flujo de respuestas.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo 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 obtener más información sobre la API, consulte [InvokeModelWithResponseStream](#) in AWS SDK for JavaScript API Reference.

Cohere Command

Converse: todos los modelos

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command mediante la API Converse de Bedrock.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Cohere Command mediante la API Converse de 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 obtener más información sobre la API, consulte [Converse](#) en la referencia de la API.AWS SDK for JavaScript

ConverseStream: Todos los modelos

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command mediante la API Converse de Bedrock y procesar el flujo de respuestas en tiempo real.

SDK para (v3 JavaScript)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Cohere Command mediante la API Converse de Bedrock y procesa el flujo de respuestas en tiempo 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 obtener más información sobre la API, consulte la referencia de [ConverseStream](#) la AWS SDK for JavaScript API.

Meta Llama

Todos los modelos: Converse API

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama mediante la API Converse de Bedrock.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Meta Llama utilizando la API Converse de 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 obtener más información sobre la API, consulta [Converse](#) en AWS SDK for JavaScript la referencia de la API.

ConverseStream: Todos los modelos

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama mediante la API Converse de Bedrock y procesar el flujo de respuestas en tiempo real.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Meta Llama utilizando la API Converse de Bedrock y procesa el flujo de respuestas en tiempo 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 obtener más información sobre la API, consulta la Referencia de [ConverseStream](#) la AWS SDK for JavaScript API.

InvokeModel: Llama 2

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama 2 mediante la API Invoke Model.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje 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 = `[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 obtener más información sobre la API, consulte [InvokeModel](#) la referencia de AWS SDK for JavaScript la API.

InvokeModel: Llama 3

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama 3 mediante la API Invoke Model.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje 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 obtener más información sobre la API, consulte [InvokeModel](#) la referencia de AWS SDK for JavaScript la API.

InvokeModelWithResponseTransmisión: Llama 2

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama 2, mediante la API Invoke Model, e imprimir el flujo de respuesta.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo 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 = `[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 obtener más información sobre la API, consulte [InvokeModelWithResponseStream](#) in AWS SDK for JavaScript API Reference.

InvokeModelWithResponseTransmisión: Llama 3

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama 3, utilizando la API Invoke Model, e imprimir el flujo de respuesta.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo 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 obtener más información sobre la API, consulte [InvokeModelWithResponseStream](#) in AWS SDK for JavaScript API Reference.

API de Mistral

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Mistral mediante la API Converse de Bedrock.

SDK para (v3) JavaScript

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Mistral utilizando la API Converse de 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 obtener más información sobre la API, consulta [Converse](#) en la referencia de la API.AWS SDK for JavaScript

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Mistral mediante la API Converse de Bedrock y cómo procesar el flujo de respuestas en tiempo real.

SDK para (v3) JavaScript

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envía un mensaje de texto a Mistral mediante la API Converse de Bedrock y procesa el flujo de respuestas en tiempo 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 obtener más información sobre la API, consulte [ConverseStream](#) la Referencia de la API. AWS SDK for JavaScript

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a los modelos Mistral mediante la API Invoke Model.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Usa la API Invoke Model para enviar un mensaje 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 = `
```

```
    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 obtener más información sobre la API, consulte [InvokeModel](#) la referencia de AWS SDK for JavaScript la API.

Escenarios

Invocar varios modelos fundacionales en Amazon Bedrock

El siguiente ejemplo de código muestra cómo preparar y enviar un mensaje a una variedad de modelos de lenguaje extendido (LLM) en Amazon Bedrock

SDK para (v3 JavaScript)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [InvokeModel](#)
 - [InvokeModelWithResponseTransmitir](#)

Ejemplos de Agents for Amazon Bedrock que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Agents for Amazon Bedrock.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Hello Agents para Amazon Bedrock

El siguiente ejemplo de código muestra cómo empezar a utilizar Agents for Amazon Bedrock.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [GetAgent](#)
 - [ListAgents](#)

Temas


- [Acciones](#)

Acciones

CreateAgent

En el siguiente ejemplo de código, se muestra cómo usar CreateAgent.

SDK para JavaScript (v3)

 Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree un agente de .

```
// 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 obtener más información sobre la API, consulta [CreateAgent](#) la Referencia AWS SDK for JavaScript de la API.

DeleteAgent

En el siguiente ejemplo de código, se muestra cómo usar DeleteAgent.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine un 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 obtener más información sobre la API, consulta [DeleteAgent](#) la Referencia AWS SDK for JavaScript de la API.

GetAgent

En el siguiente ejemplo de código, se muestra cómo usar GetAgent.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga un 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 obtener más información sobre la API, consulta [GetAgent](#) la Referencia AWS SDK for JavaScript de la API.

ListAgentActionGroups

En el siguiente ejemplo de código, se muestra cómo usar `ListAgentActionGroups`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los grupos de acciones de un 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 obtener más información sobre la API, consulta [ListAgentActionGroups](#) la Referencia AWS SDK for JavaScript de la API.

ListAgents

En el siguiente ejemplo de código, se muestra cómo usar `ListAgents`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los agentes que pertenecen a una cuenta.

```
// 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 obtener más información sobre la API, consulta [ListAgents](#) la Referencia AWS SDK for JavaScript de la API.

Ejemplos de Agents for Amazon Bedrock Runtime que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la versión AWS SDK for JavaScript 3 con Agents for Amazon Bedrock Runtime.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

InvokeAgent

En el siguiente ejemplo de código, se muestra cómo usar InvokeAgent.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [InvokeAgent](#) la Referencia AWS SDK for JavaScript de la API.

CloudWatch ejemplos de uso del SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con CloudWatch

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

DeleteAlarms

En el siguiente ejemplo de código, se muestra cómo usar DeleteAlarms.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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.
  });
  client.send(command);
}
```



```
});

try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```


Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteAlarms](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteAlarms](#) la Referencia AWS SDK for JavaScript de la API.

DescribeAlarmsForMetric

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAlarmsForMetric`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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();
```

Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";  
  
export const client = new CloudWatchClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DescribeAlarmsForMetric](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DescribeAlarmsForMetric](#) la Referencia AWS SDK for JavaScript de la API.

DisableAlarmActions

En el siguiente ejemplo de código, se muestra cómo usar `DisableAlarmActions`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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();
```

Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [DisableAlarmsección Acciones](#) en la referencia de la AWS SDK for JavaScript API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).

- Para obtener más información sobre la API, consulta la [DisableAlarmsección Acciones](#) en la referencia de la AWS SDK for JavaScript API.

EnableAlarmActions

En el siguiente ejemplo de código, se muestra cómo usar `EnableAlarmActions`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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();
```

Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [EnableAlarmsección Acciones](#) en la referencia de la AWS SDK for JavaScript API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [EnableAlarmsección Acciones](#) en la referencia de la AWS SDK for JavaScript API.

ListMetrics

En el siguiente ejemplo de código, se muestra cómo usar `ListMetrics`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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);
};
```

Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListMetrics](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListMetrics](#) la Referencia AWS SDK for JavaScript de la API.

PutMetricAlarm

En el siguiente ejemplo de código, se muestra cómo usar PutMetricAlarm.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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();
```

Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutMetricAlarm](#) in AWS SDK for JavaScript API Reference.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutMetricAlarm](#) in AWS SDK for JavaScript API Reference.

PutMetricData

En el siguiente ejemplo de código, se muestra cómo usar PutMetricData.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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();
```


Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutMetriclos datos](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutMetricData](#) en la referencia AWS SDK for JavaScript de la API.

CloudWatch Ejemplos de eventos que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la versión AWS SDK for JavaScript (v3) con CloudWatch eventos.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

PutEvents

En el siguiente ejemplo de código, se muestra cómo usar PutEvents.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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();
```

Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutEvents](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutEvents](#) la Referencia AWS SDK for JavaScript de la API.

PutRule

En el siguiente ejemplo de código, se muestra cómo usar PutRule.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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();
```

Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutRule](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutRule](#) la Referencia AWS SDK for JavaScript de la API.

PutTargets

En el siguiente ejemplo de código, se muestra cómo usar PutTargets.

SDK para JavaScript (v3)

 Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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();
```

Cree el cliente en un módulo separado y expórtelo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";
```

```
export const client = new CloudWatchEventsClient({});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutTargets](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutTargets](#) la Referencia AWS SDK for JavaScript de la API.

CloudWatch Registra ejemplos con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la AWS SDK for JavaScript versión 3 con CloudWatch registros.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

CreateLogGroup

En el siguiente ejemplo de código, se muestra cómo usar CreateLogGroup.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateLogAgrupar](#) en la referencia de la AWS SDK for JavaScript API.

DeleteLogGroup

En el siguiente ejemplo de código, se muestra cómo usar DeleteLogGroup.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteLogAgrupar](#) en la referencia de la AWS SDK for JavaScript API.

DeleteSubscriptionFilter

En el siguiente ejemplo de código, se muestra cómo usar DeleteSubscriptionFilter.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la sección [DeleteSubscriptionFiltrar](#) en la referencia de la AWS SDK for JavaScript API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });  
  
var params = {  
  filterName: "FILTER",  
  logGroupName: "LOG_GROUP",  
};  
  
cw1.deleteSubscriptionFilter(params, function (err, data) {  
  if (err) {  
    console.log("Error", err);  
  } else {  
    console.log("Success", data);  
  }  
});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la sección [DeleteSubscriptionFiltrar](#) en la referencia de la AWS SDK for JavaScript API.

DescribeLogGroups

En el siguiente ejemplo de código, se muestra cómo usar DescribeLogGroups.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [DescribeLogsección Grupos](#) en la referencia de la AWS SDK for JavaScript API.

DescribeSubscriptionFilters

En el siguiente ejemplo de código, se muestra cómo usar DescribeSubscriptionFilters.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [DescribeSubscriptionsección Filtros](#) en la referencia de la AWS SDK for JavaScript API.

SDK para JavaScript (v2)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [DescribeSubscriptionsección Filtros](#) en la referencia de la AWS SDK for JavaScript API.

GetQueryResults

En el siguiente ejemplo de código, se muestra cómo usar `GetQueryResults`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
```

```
* @param {string} queryId
*/
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
```

- Para obtener más información sobre la API, consulta [GetQuery los resultados](#) en la referencia AWS SDK for JavaScript de la API.

PutSubscriptionFilter

En el siguiente ejemplo de código, se muestra cómo usar PutSubscriptionFilter.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la sección [PutSubscriptionFiltrar](#) en la referencia de la AWS SDK for JavaScript API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cw1.putSubscriptionFilter(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la sección [PutSubscriptionFiltrar](#) en la referencia de la AWS SDK for JavaScript API.

StartLiveTail

En el siguiente ejemplo de código, se muestra cómo usar `StartLiveTail`.

SDK para JavaScript (v3)

Incluir los archivos requeridos.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-cloudwatch-logs";
```

Gestione los eventos de la sesión de 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 la sesión de 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);
}
```

Detenga la sesión de Live Tail una vez transcurrido un periodo de tiempo.


```
/* Set a timeout to close the client. This will stop the Live Tail session. */
setTimeout(function() {
  console.log("Client timeout");
  client.destroy();
}, 10000);
```

- Para obtener más información sobre la API, consulta [StartLiveTail referencia sobre la AWS SDK for JavaScript API de Tail](#) in.

StartQuery

En el siguiente ejemplo de código, se muestra cómo usar StartQuery.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/**
 * 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 obtener más información sobre la API, consulta [StartQuery](#) la Referencia AWS SDK for JavaScript de la API.

Escenarios

Ejecución de una consulta de gran tamaño

El siguiente ejemplo de código muestra cómo usar CloudWatch los registros para consultar más de 10 000 registros.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este es el punto 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}`,
```

```
);
```

Esta es una clase que divide las consultas en varios pasos si es necesario.

```
// 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 obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for JavaScript .
 - [GetQueryResultados](#)
 - [StartQuery](#)

CodeBuild ejemplos de uso del SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con CodeBuild

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

CreateProject

En el siguiente ejemplo de código, se muestra cómo usar CreateProject.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree un proyecto.

```
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-xxxxxx-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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateProject](#) la Referencia AWS SDK for JavaScript de la API.

Ejemplos de proveedores de identidad de Amazon Cognito que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Cognito Identity Provider.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon Cognito

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon Cognito.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre las API, consulta la [ListUsersección Grupos](#) en la referencia de las AWS SDK for JavaScript API.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

AdminGetUser

En el siguiente ejemplo de código, se muestra cómo usar AdminGetUser.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
    Username: username,
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta la referencia [AdminGetsobre el usuario](#) en la AWS SDK for JavaScript API.

AdminInitiateAuth

En el siguiente ejemplo de código, se muestra cómo usar AdminInitiateAuth.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [AdminInitiateAuth en la](#) referencia de la AWS SDK for JavaScript API.

AdminRespondToAuthChallenge

En el siguiente ejemplo de código, se muestra cómo usar `AdminRespondToAuthChallenge`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [AdminRespondToAuthChallenge](#) in AWS SDK for JavaScript API Reference.

AssociateSoftwareToken

En el siguiente ejemplo de código, se muestra cómo usar AssociateSoftwareToken.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const associateSoftwareToken = (session) => {  
    const client = new CognitoIdentityProviderClient({});  
    const command = new AssociateSoftwareTokenCommand({  
        Session: session,  
    });  
  
    return client.send(command);  
};
```

- Para obtener más información sobre la API, consulta el [AssociateSoftwareToken](#) en la referencia de la AWS SDK for JavaScript API.

ConfirmDevice

En el siguiente ejemplo de código, se muestra cómo usar ConfirmDevice.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [ConfirmDevice](#) la Referencia AWS SDK for JavaScript de la API.

ConfirmSignUp

En el siguiente ejemplo de código, se muestra cómo usar ConfirmSignUp.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const confirmSignUp = ({ clientId, username, code }) => {
```

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

const command = new ConfirmSignUpCommand({
  ClientId: clientId,
  Username: username,
  ConfirmationCode: code,
});

return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [ConfirmSignla sección Up](#) in AWS SDK for JavaScript API Reference.

InitiateAuth

En el siguiente ejemplo de código, se muestra cómo usar InitiateAuth.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [InitiateAuth](#) la Referencia AWS SDK for JavaScript de la API.

ListUsers

En el siguiente ejemplo de código, se muestra cómo usar ListUsers.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [ListUsers](#) la Referencia AWS SDK for JavaScript de la API.

ResendConfirmationCode

En el siguiente ejemplo de código, se muestra cómo usar ResendConfirmationCode.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta el [ResendConfirmationcódigo](#) en la referencia AWS SDK for JavaScript de la API.

RespondToAuthChallenge

En el siguiente ejemplo de código, se muestra cómo usar RespondToAuthChallenge.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [RespondToAuthChallenge](#) la Referencia AWS SDK for JavaScript de la API.

SignUp

En el siguiente ejemplo de código, se muestra cómo usar SignUp.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [SignUp](#) la Referencia AWS SDK for JavaScript de la API.

VerifySoftwareToken

En el siguiente ejemplo de código, se muestra cómo usar `VerifySoftwareToken`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta el [VerifySoftwareartículo Token](#) en la referencia de la AWS SDK for JavaScript API.

Escenarios

Registro de un usuario en un grupo de usuarios que requiera MFA

En el siguiente ejemplo de código, se muestra cómo:

- Registre y confirme a un usuario con un nombre de usuario, una contraseña y una dirección de correo electrónico.
- Configure la autenticación multifactor asociando una aplicación MFA al usuario.
- Inicie sesión con una contraseña y un código MFA.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Para obtener la mejor experiencia, clone el GitHub repositorio y ejecute este ejemplo. El código siguiente es una muestra de la aplicación de ejemplo 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 [userId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userId,
      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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [AdminGetUsuario](#)
 - [AdminInitiateAutenticación](#)
 - [AdminRespondToAuthDesafío](#)
 - [AssociateSoftwareSímbolo](#)
 - [ConfirmDevice](#)
 - [ConfirmSignArriba](#)

- [InitiateAuth](#)
- [ListUsers](#)
- [ResendConfirmationCódigo](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

Ejemplos de Amazon DocumentDB que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon DocumentDB.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Ejemplos sin servidor](#)

Ejemplos sin servidor

Invocar una función Lambda desde un activador de Amazon DocumentDB

El siguiente ejemplo de código muestra cómo implementar una función de Lambda que recibe un evento que se desencadena al recibir registros de un flujo de cambios de DocumentDB. La función recupera la carga útil de DocumentDB y registra el contenido del registro.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de Amazon DocumentDB con Lambda mediante. 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));
};
```

Ejemplos de DynamoDB que utilizan el SDK JavaScript para (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con DynamoDB.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a DynamoDB

En los siguientes ejemplos de código, se muestra cómo empezar a utilizar DynamoDB.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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.TableNames.join("\n"));
  return response;
};
```

- Para obtener más información sobre la API, consulta [ListTables](#) la Referencia AWS SDK for JavaScript de la API.

Temas

- [Acciones](#)
- [Escenarios](#)
- [Ejemplos sin servidor](#)

Acciones

BatchExecuteStatement

En el siguiente ejemplo de código, se muestra cómo usar BatchExecuteStatement.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Crear un lote de elementos con 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;
};
```

Obtener un lote de elementos con 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;
};
```

Actualizar un lote de elementos con 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;
};
```

Eliminar un lote de elementos con 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 obtener más información sobre la API, consulta la [BatchExecutedeclaración](#) en la referencia AWS SDK for JavaScript de la API.

BatchGetItem

En el siguiente ejemplo de código, se muestra cómo usar BatchGetItem.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener detalles sobre la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener información sobre la API, consulte el [BatchGetartículo](#) de la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener detalles sobre la API, consulta el [BatchGet artículo](#) de la referencia AWS SDK for JavaScript de la API.

BatchWriteItem

En el siguiente ejemplo de código, se muestra cómo usar BatchWriteItem.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener detalles sobre la 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 obtener información sobre la API, consulte el [BatchWriteartículo](#) de la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener detalles sobre la API, consulta el [BatchWrite artículo](#) de la referencia AWS SDK for JavaScript de la API.

CreateTable

En el siguiente ejemplo de código, se muestra cómo usar CreateTable.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateTable](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateTable](#) la Referencia AWS SDK for JavaScript de la API.

DeleteItem

En el siguiente ejemplo de código, se muestra cómo usar DeleteItem.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener detalles sobre la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulte [DeleteItem](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Eliminar un elemento de una tabla.

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

Eliminar un elemento de una tabla con el cliente de documentos de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteItem](#) la Referencia AWS SDK for JavaScript de la API.

DeleteTable

En el siguiente ejemplo de código, se muestra cómo usar DeleteTable.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteTable](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteTable](#) la Referencia AWS SDK for JavaScript de la API.

DescribeTable

En el siguiente ejemplo de código, se muestra cómo usar DescribeTable.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DescribeTable](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DescribeTable](#) la Referencia AWS SDK for JavaScript de la API.

DescribeTimeToLive

En el siguiente ejemplo de código, se muestra cómo 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 obtener más información sobre la API, consulte [DescribeTimeToLive](#) la Referencia de AWS SDK for JavaScript la API.

ExecuteStatement

En el siguiente ejemplo de código, se muestra cómo usar ExecuteStatement.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Crear un elemento con 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;
};
```

Obtener un elemento con 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;
};
```

Actualizar un elemento con 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;
};
```

Eliminar un elemento con 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 obtener más información sobre la API, consulta [ExecuteStatement](#) la Referencia AWS SDK for JavaScript de la API.

GetItem

En el siguiente ejemplo de código, se muestra cómo usar GetItem.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener detalles sobre la 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 obtener más información sobre la API, consulte [GetItem](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtener un elemento de una tabla.

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

Obtener un elemento de una tabla con el cliente de documentos de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetItem](#) la Referencia AWS SDK for JavaScript de la API.

ListTables

En el siguiente ejemplo de código, se muestra cómo usar `ListTables`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListTables](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListTables](#) la Referencia AWS SDK for JavaScript de la API.

PutItem

En el siguiente ejemplo de código, se muestra cómo usar PutItem.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener detalles sobre la 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 obtener más información sobre la API, consulte [PutItem](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Colocar un elemento en una tabla.

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

Colocar un elemento en una tabla con el cliente de documentos de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutItem](#) la Referencia AWS SDK for JavaScript de la API.

Query

En el siguiente ejemplo de código, se muestra cómo usar Query.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener detalles sobre la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener información sobre la API, consulte [Query](#) en la referencia de la API de AWS SDK for JavaScript .

SDK para JavaScript (v2)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener información sobre la API, consulte [Query](#) en la referencia de la API de AWS SDK for JavaScript .

Scan

En el siguiente ejemplo de código, se muestra cómo usar Scan.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener detalles sobre la 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 obtener información sobre la API, consulte [Scan](#) en la referencia de la API de AWS SDK for JavaScript .

SDK para JavaScript (v2)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener información sobre la API, consulte [Scan](#) en la referencia de la API de AWS SDK for JavaScript .

UpdateItem

En el siguiente ejemplo de código, se muestra cómo usar `UpdateItem`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener detalles sobre la 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 obtener más información sobre la API, consulte [UpdateItem](#) la Referencia AWS SDK for JavaScript de la API.

UpdateTimeToLive

En el siguiente ejemplo de código, se muestra cómo usar UpdateTimeToLive.

SDK para JavaScript (v3)

Habilite el TTL en una tabla de DynamoDB existente.

```
// 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');
```

Deshabilite el TTL en una tabla de DynamoDB existente.

```
// 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 obtener más información sobre la API, consulte la Referencia de [UpdateTimeToLive](#) la AWS SDK for JavaScript API.

Escenarios

Actualiza condicionalmente el TTL de un elemento

El siguiente ejemplo de código muestra cómo actualizar condicionalmente el TTL de un elemento.

SDK para JavaScript (v3)

Actualice el TTL en un elemento de DynamoDB existente en una tabla, con una condición.

```
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 obtener más información sobre la API, consulte la Referencia de [UpdateItem](#) la AWS SDK for JavaScript API.

Cree un elemento con un TTL

El siguiente ejemplo de código muestra cómo crear un elemento con 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 obtener más información sobre la API, consulte [PutItem](#) la Referencia de AWS SDK for JavaScript la API.

Introducción a tablas, elementos y consultas

En el siguiente ejemplo de código, se muestra cómo:

- Creación de una tabla que pueda contener datos de películas.
- Colocar, obtener y actualizar una sola película en la tabla.
- Escribir los datos de películas en la tabla a partir de un archivo JSON de ejemplo.
- Consultar películas que se hayan estrenado en un año determinado.
- Buscar películas que se hayan estrenado en un intervalo de años.
- Eliminación de una película de la tabla y, a continuación, eliminar la tabla.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [BatchWriteArtículo](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)


- [Query](#)
- [Scan](#)
- [UpdateItem](#)

Consultar una tabla mediante lotes de instrucciones PartiQL

En el siguiente ejemplo de código, se muestra cómo:

- Obtención de un lote de elementos mediante la ejecución de varias instrucciones SELECT.
- Agregar un lote de elementos mediante la ejecución de varias instrucciones INSERT.
- Actualizar un lote de elementos con la ejecución de varias instrucciones UPDATE.
- Eliminación de un lote de elementos con la ejecución de varias instrucciones DELETE.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar instrucciones PartiQL por lotes.

```
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 addItemStatementCommand = 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 obtener más información sobre la API, consulta la [BatchExecutedeclaración](#) en la referencia AWS SDK for JavaScript de la API.

Consultar una tabla con PartiQL

En el siguiente ejemplo de código, se muestra cómo:

- Obtención de un artículo mediante una instrucción SELECT.
- Agregar un elemento mediante una instrucción INSERT.
- Actualizar un elemento mediante una instrucción UPDATE.
- Eliminación de un elemento mediante una instrucción DELETE.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar instrucciones PartiQL individuales.

```
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 obtener más información sobre la API, consulta [ExecuteStatement](#) la Referencia AWS SDK for JavaScript de la API.

Consulta los elementos TTL

El siguiente ejemplo de código muestra cómo consultar elementos 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 obtener información sobre la API, consulte [Query](#) en la referencia de la API de AWS SDK for JavaScript .

Actualizar el TTL de un elemento

El siguiente ejemplo de código muestra cómo actualizar el TTL de un artículo.

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 obtener más información sobre la API, consulte [UpdateItem](#) la Referencia de AWS SDK for JavaScript la API.

Ejemplos sin servidor

Invocación de una función de Lambda desde un desencadenador de DynamoDB

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento que se desencadena al recibir registros de una transmisión de DynamoDB. La función recupera la carga útil de DynamoDB y registra el contenido del registro.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de DynamoDB con Lambda mediante JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
};

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consumir un evento de DynamoDB con Lambda mediante 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)}`);  
};
```

Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB

El siguiente ejemplo de código muestra cómo implementar una respuesta por lotes parcial para las funciones de Lambda que reciben eventos de una transmisión de DynamoDB. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Cómo informar de errores en los elementos de lote de DynamoDB con Lambda mediante 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: [] };  
};
```

Cómo informar de errores en los elementos de lote de DynamoDB con Lambda mediante 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  
}
```

Ejemplos de Amazon EC2 con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon EC2.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon EC2

En los siguientes ejemplos de código, se muestra cómo empezar a utilizar Amazon EC2.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [DescribeSecuritysección Grupos](#) en la referencia de la AWS SDK for JavaScript API.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

AllocateAddress

En el siguiente ejemplo de código, se muestra cómo usar AllocateAddress.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [AllocateAddress](#) la Referencia AWS SDK for JavaScript de la API.

AssociateAddress

En el siguiente ejemplo de código, se muestra cómo usar AssociateAddress.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [AssociateAddress](#) la Referencia AWS SDK for JavaScript de la API.

AuthorizeSecurityGroupIngress

En el siguiente ejemplo de código, se muestra cómo usar `AuthorizeSecurityGroupIngress`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [AuthorizeSecurityGroupIngress](#) la Referencia AWS SDK for JavaScript de la API.

CreateKeyPair

En el siguiente ejemplo de código, se muestra cómo usar CreateKeyPair.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [CreateKeysección Pair](#) in AWS SDK for JavaScript API Reference.

CreateLaunchTemplate

En el siguiente ejemplo de código, se muestra cómo usar CreateLaunchTemplate.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [CreateLaunchplantilla](#) en la referencia AWS SDK for JavaScript de la API.

CreateSecurityGroup

En el siguiente ejemplo de código, se muestra cómo usar CreateSecurityGroup.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateSecurityAgrupar](#) en la referencia de la AWS SDK for JavaScript API.

DeleteKeyPair

En el siguiente ejemplo de código, se muestra cómo usar DeleteKeyPair.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [DeleteKeysección Pair](#) in AWS SDK for JavaScript API Reference.

DeleteLaunchTemplate

En el siguiente ejemplo de código, se muestra cómo usar DeleteLaunchTemplate.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
await client.send(  
  new DeleteLaunchTemplateCommand({  
    LaunchTemplateName: NAMES.launchTemplateName,  
  }),  
);
```

- Para obtener más información sobre la API, consulta la [DeleteLaunchplantilla](#) en la referencia AWS SDK for JavaScript de la API.

DeleteSecurityGroup

En el siguiente ejemplo de código, se muestra cómo usar DeleteSecurityGroup.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteSecurityAgrupar](#) en la referencia de la AWS SDK for JavaScript API.

DescribeAddresses

En el siguiente ejemplo de código, se muestra cómo usar DescribeAddresses.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DescribeAddresses](#) la Referencia AWS SDK for JavaScript de la API.

DescribeIamInstanceProfileAssociations

En el siguiente ejemplo de código, se muestra cómo usar `DescribeIamInstanceProfileAssociations`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
```

- Para obtener más información sobre la API, consulta la [DescribelamInstanceProfilesección Asociaciones](#) en la referencia de la AWS SDK for JavaScript API.

DescribeImages

En el siguiente ejemplo de código, se muestra cómo usar `DescribeImages`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DescribeImages](#) la Referencia AWS SDK for JavaScript de la API.

DescribeInstanceTypes

En el siguiente ejemplo de código, se muestra cómo usar DescribeInstanceTypes.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [DescribeInstances sección Tipos](#) en la referencia de la AWS SDK for JavaScript API.

DescribeInstances

En el siguiente ejemplo de código, se muestra cómo usar DescribeInstances.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DescribeInstances](#) la Referencia AWS SDK for JavaScript de la API.

DescribeKeyPairs

En el siguiente ejemplo de código, se muestra cómo usar DescribeKeyPairs.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [DescribeKeysección Pairs](#) in AWS SDK for JavaScript API Reference.

DescribeRegions

En el siguiente ejemplo de código, se muestra cómo usar DescribeRegions.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DescribeRegions](#) la Referencia AWS SDK for JavaScript de la API.

DescribeSecurityGroups

En el siguiente ejemplo de código, se muestra cómo usar DescribeSecurityGroups.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la [DescribeSecuritysección Grupos](#) en la referencia de la AWS SDK for JavaScript API.

DescribeSubnets

En el siguiente ejemplo de código, se muestra cómo usar DescribeSubnets.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DescribeSubnets](#) la Referencia AWS SDK for JavaScript de la API.

DescribeVpcs

En el siguiente ejemplo de código, se muestra cómo usar DescribeVpcs.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```



```
);
```

- Para obtener más información sobre la API, consulta [DescribeVpcs](#) la Referencia AWS SDK for JavaScript de la API.

DisassociateAddress

En el siguiente ejemplo de código, se muestra cómo usar `DisassociateAddress`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DisassociateAddress](#) la Referencia AWS SDK for JavaScript de la API.

MonitorInstances

En el siguiente ejemplo de código, se muestra cómo usar MonitorInstances.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [MonitorInstances](#) la Referencia AWS SDK for JavaScript de la API.

RebootInstances

En el siguiente ejemplo de código, se muestra cómo usar `RebootInstances`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [RebootInstances](#) la Referencia AWS SDK for JavaScript de la API.

ReleaseAddress

En el siguiente ejemplo de código, se muestra cómo usar `ReleaseAddress`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [ReleaseAddress](#) la Referencia AWS SDK for JavaScript de la API.

ReplaceIamInstanceProfileAssociation

En el siguiente ejemplo de código, se muestra cómo usar `ReplaceIamInstanceProfileAssociation`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
);
```

- Para obtener más información sobre la API, consulta [ReplacelamInstanceProfilela sección Asociación](#) en la referencia de la AWS SDK for JavaScript API.

RunInstances

En el siguiente ejemplo de código, se muestra cómo usar RunInstances.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [RunInstances](#) la Referencia AWS SDK for JavaScript de la API.

StartInstances

En el siguiente ejemplo de código, se muestra cómo usar StartInstances.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [StartInstances](#) la Referencia AWS SDK for JavaScript de la API.

StopInstances

En el siguiente ejemplo de código, se muestra cómo usar StopInstances.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [StopInstances](#) la Referencia AWS SDK for JavaScript de la API.

TerminateInstances

En el siguiente ejemplo de código, se muestra cómo usar TerminateInstances.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [TerminateInstances](#) la Referencia AWS SDK for JavaScript de la API.

UnmonitorInstances

En el siguiente ejemplo de código, se muestra cómo usar `UnmonitorInstances`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [UnmonitorInstances](#) la Referencia AWS SDK for JavaScript de la API.


Escenarios

Creación y administración de un servicio resiliente

El siguiente ejemplo de código muestra cómo crear un servicio web con equilibrio de carga que muestre recomendaciones de libros, películas y canciones. El ejemplo muestra cómo responde el servicio a los errores y cómo reestructurarlo para aumentar la resiliencia cuando se produzcan errores.

- Utilice un grupo de Amazon EC2 Auto Scaling para crear instancias de Amazon Elastic Compute Cloud (Amazon EC2) basadas en una plantilla de lanzamiento y para mantener el número de instancias dentro de un rango específico.
- Administre y distribuya las solicitudes HTTP con Elastic Load Balancing.
- Supervise el estado de las instancias de un grupo de escalado automático y reenvíe las solicitudes solo a las instancias en buen estado.
- Ejecute un servidor web Python en cada instancia de EC2 para administrar las solicitudes HTTP. El servidor web responde con recomendaciones y comprobaciones de estado.
- Simule un servicio de recomendaciones con una tabla de Amazon DynamoDB.
- Controle la respuesta del servidor web a las solicitudes y las comprobaciones de estado actualizando AWS Systems Manager los parámetros.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecute el escenario interactivo en un símbolo del sistema.

```
#!/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 escenarios = {
  // 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);
}
```

Cree los pasos para implementar todos los 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,
    );
  }
}),
];

```

Cree los pasos para ejecutar la demostración.

```

// 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;
}

```

Cree los pasos para destruir todos los 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 obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for JavaScript .
 - [AttachLoadBalancerTargetGrupos](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstancePerfil](#)
 - [CreateLaunchPlantilla](#)
 - [CreateListener](#)
 - [CreateLoadEquilibrador](#)
 - [CreateTargetGrupo](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstancePerfil](#)
 - [DeleteLaunchPlantilla](#)
 - [DeleteLoadEquilibrador](#)
 - [DeleteTargetGrupo](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZonas](#)
 - [DescribeIamInstanceProfileAsociaciones](#)
 - [DescribeInstances](#)
 - [DescribeLoadEquilibradores](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGrupos](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)


- [RebootInstances](#)
- [ReplacelamInstanceProfileAsociación](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Comience a utilizar instancias

En el siguiente ejemplo de código, se muestra cómo:

- Cree un par de claves y un grupo de seguridad.
- Seleccione una Imagen de máquina de Amazon (AMI) y un tipo de instancia; a continuación, cree una instancia.
- Detenga y vuelva a iniciar la instancia.
- Asocie una dirección IP elástica a su instancia.
- Conéctese a tu instancia con SSH y, a continuación, limpie los recursos.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar un escenario interactivo en un símbolo del sistema.

```
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(
    `
You should now be able to SSH into your instance from another terminal`,
    `
${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(
    `
Instance started. The IP address changed from
${instanceDetails.PublicIpAddress} to ${ipAddressAfterRestart}`,
    `
${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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [AllocateAddress](#)
 - [AssociateAddress](#)

- [AuthorizeSecurityGroupIngress](#)
- [CreateKeyPareja](#)
- [CreateSecurityGrupo](#)
- [DeleteKeyPareja](#)
- [DeleteSecurityGrupo](#)
- [DescribeImages](#)
- [DescribeInstanceTipos](#)
- [DescribeInstances](#)
- [DescribeKeyPares](#)
- [DescribeSecurityGrupos](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

Ejemplos de Elastic Load Balancing: versión 2 con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Elastic Load Balancing: versión 2.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo **configurar y ejecutar el código en su contexto.**

Introducción

Introducción al equilibrador de carga elástico

En los siguientes ejemplos de código se muestra cómo empezar a utilizar un equilibrador de carga elástico.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [DescribeLoadBalancers](#) en la referencia de AWS SDK for JavaScript la API.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

CreateListener

En el siguiente ejemplo de código, se muestra cómo usar CreateListener.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateListener](#) la Referencia AWS SDK for JavaScript de la API.

CreateLoadBalancer

En el siguiente ejemplo de código, se muestra cómo usar `CreateLoadBalancer`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateLoadBalancer](#) en la referencia de AWS SDK for JavaScript la API.

CreateTargetGroup

En el siguiente ejemplo de código, se muestra cómo usar `CreateTargetGroup`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateTargetAgrupar](#) en la referencia de la AWS SDK for JavaScript API.

DeleteLoadBalancer

En el siguiente ejemplo de código, se muestra cómo usar DeleteLoadBalancer.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteLoadBalancer](#) en la referencia de AWS SDK for JavaScript la API.

DeleteTargetGroup

En el siguiente ejemplo de código, se muestra cómo usar DeleteTargetGroup.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteTargetAgrupar](#) en la referencia de la AWS SDK for JavaScript API.

DescribeLoadBalancers

En el siguiente ejemplo de código, se muestra cómo usar DescribeLoadBalancers.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [DescribeLoadBalancers](#) en la referencia de AWS SDK for JavaScript la API.

DescribeTargetGroups

En el siguiente ejemplo de código, se muestra cómo usar DescribeTargetGroups.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- Para obtener más información sobre la API, consulta la [DescribeTargetsección Grupos](#) en la referencia de la AWS SDK for JavaScript API.

DescribeTargetHealth

En el siguiente ejemplo de código, se muestra cómo usar DescribeTargetHealth.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- Para obtener más información sobre la API, consulta [DescribeTargetHealth](#) in AWS SDK for JavaScript API Reference.


Escenarios

Creación y administración de un servicio resiliente

El siguiente ejemplo de código muestra cómo crear un servicio web con equilibrio de carga que muestre recomendaciones de libros, películas y canciones. El ejemplo muestra cómo responde el servicio a los errores y cómo reestructurarlo para aumentar la resiliencia cuando se produzcan errores.

- Utilice un grupo de Amazon EC2 Auto Scaling para crear instancias de Amazon Elastic Compute Cloud (Amazon EC2) basadas en una plantilla de lanzamiento y para mantener el número de instancias dentro de un rango específico.
- Administre y distribuya las solicitudes HTTP con Elastic Load Balancing.
- Supervise el estado de las instancias de un grupo de escalado automático y reenvíe las solicitudes solo a las instancias en buen estado.
- Ejecute un servidor web Python en cada instancia de EC2 para administrar las solicitudes HTTP. El servidor web responde con recomendaciones y comprobaciones de estado.
- Simule un servicio de recomendaciones con una tabla de Amazon DynamoDB.
- Controle la respuesta del servidor web a las solicitudes y las comprobaciones de estado actualizando AWS Systems Manager los parámetros.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecute el escenario interactivo en un símbolo del sistema.

```
#!/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 escenarios = {
  // 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);
}
```

Cree los pasos para implementar todos los 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,
    );
  }
}),
];
```

Cree los pasos para ejecutar la demostración.

```
// 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;
}

```

Cree los pasos para destruir todos los 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 obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for JavaScript .
 - [AttachLoadBalancerTargetGrupos](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstancePerfil](#)
 - [CreateLaunchPlantilla](#)
 - [CreateListener](#)
 - [CreateLoadEquilibrador](#)
 - [CreateTargetGrupo](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstancePerfil](#)
 - [DeleteLaunchPlantilla](#)
 - [DeleteLoadEquilibrador](#)
 - [DeleteTargetGrupo](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZonas](#)
 - [DescribeIamInstanceProfileAsociaciones](#)
 - [DescribeInstances](#)
 - [DescribeLoadEquilibradores](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGrupos](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)

- [RebootInstances](#)
- [ReplacelamInstanceProfileAsociación](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

EventBridge ejemplos de uso del SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con EventBridge

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

PutEvents

En el siguiente ejemplo de código, se muestra cómo usar PutEvents.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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 obtener más información sobre la API, consulta [PutEvents](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 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 obtener más información sobre la API, consulta [PutEvents](#) la Referencia AWS SDK for JavaScript de la API.

PutRule

En el siguiente ejemplo de código, se muestra cómo usar PutRule.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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 obtener más información sobre la API, consulta [PutRule](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 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 obtener más información sobre la API, consulta [PutRule](#) la Referencia AWS SDK for JavaScript de la API.

PutTargets

En el siguiente ejemplo de código, se muestra cómo usar PutTargets.

SDK para JavaScript (v3)

Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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 obtener más información sobre la API, consulta [PutTargets](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 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 obtener más información sobre la API, consulta [PutTargets](#) la Referencia AWS SDK for JavaScript de la API.

AWS Glue ejemplos de uso del SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con AWS Glue

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola AWS Glue

En los siguientes ejemplos de código se muestra cómo empezar a utilizar AWS Glue.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [ListJobs](#) la Referencia AWS SDK for JavaScript de la API.

Temas


- [Acciones](#)
- [Escenarios](#)

Acciones

CreateCrawler

En el siguiente ejemplo de código, se muestra cómo usar CreateCrawler.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateCrawler](#) la Referencia AWS SDK for JavaScript de la API.

CreateJob

En el siguiente ejemplo de código, se muestra cómo usar CreateJob.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateJob](#) la Referencia AWS SDK for JavaScript de la API.

DeleteCrawler

En el siguiente ejemplo de código, se muestra cómo usar DeleteCrawler.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

```
};
```

- Para obtener más información sobre la API, consulta [DeleteCrawler](#) la Referencia AWS SDK for JavaScript de la API.

DeleteDatabase

En el siguiente ejemplo de código, se muestra cómo usar DeleteDatabase.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const deleteDatabase = (databaseName) => {  
  const client = new GlueClient({});  
  
  const command = new DeleteDatabaseCommand({  
    Name: databaseName,  
  });  
  
  return client.send(command);  
};
```

- Para obtener más información sobre la API, consulta [DeleteDatabase](#) la Referencia AWS SDK for JavaScript de la API.

DeleteJob

En el siguiente ejemplo de código, se muestra cómo usar DeleteJob.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [DeleteJob](#) la Referencia AWS SDK for JavaScript de la API.

DeleteTable

En el siguiente ejemplo de código, se muestra cómo usar DeleteTable.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
```

```
    Name: tableName,
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [DeleteTable](#) la Referencia AWS SDK for JavaScript de la API.

GetCrawler

En el siguiente ejemplo de código, se muestra cómo usar `GetCrawler`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [GetCrawler](#) la Referencia AWS SDK for JavaScript de la API.

GetDatabase

En el siguiente ejemplo de código, se muestra cómo usar `GetDatabase`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [GetDatabases](#) la Referencia AWS SDK for JavaScript de la API.

GetDatabases

En el siguiente ejemplo de código, se muestra cómo usar GetDatabases.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});
```



```
    return client.send(command);  
};
```

- Para obtener más información sobre la API, consulta [GetDatabases](#) la Referencia AWS SDK for JavaScript de la API.

GetJob

En el siguiente ejemplo de código, se muestra cómo usar GetJob.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getJob = (jobName) => {  
    const client = new GlueClient({});  
  
    const command = new GetJobCommand({  
        JobName: jobName,  
    });  
  
    return client.send(command);  
};
```

- Para obtener más información sobre la API, consulta [GetJob](#) la Referencia AWS SDK for JavaScript de la API.

GetJobRun

En el siguiente ejemplo de código, se muestra cómo usar GetJobRun.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta la referencia sobre cómo [GetJobejecutar](#) en la AWS SDK for JavaScript API.

GetJobRuns

En el siguiente ejemplo de código, se muestra cómo usar GetJobRuns.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });
};
```

```
    return client.send(command);  
};
```

- Para obtener más información sobre la API, consulta [GetJob AWS SDK for JavaScript Runs](#) in API Reference.

GetTables

En el siguiente ejemplo de código, se muestra cómo usar `GetTables`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getTables = (databaseName) => {  
    const client = new GlueClient({});  
  
    const command = new GetTablesCommand({  
        DatabaseName: databaseName,  
    });  
  
    return client.send(command);  
};
```

- Para obtener más información sobre la API, consulta [GetTables](#) la Referencia AWS SDK for JavaScript de la API.

ListJobs

En el siguiente ejemplo de código, se muestra cómo usar `ListJobs`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [ListJobs](#) la Referencia AWS SDK for JavaScript de la API.

StartCrawler

En el siguiente ejemplo de código, se muestra cómo usar StartCrawler.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });
};
```

```
    return client.send(command);
  };
```

- Para obtener más información sobre la API, consulta [StartCrawler](#) la Referencia AWS SDK for JavaScript de la API.

StartJobRun

En el siguiente ejemplo de código, se muestra cómo usar StartJobRun.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la referencia sobre cómo [StartJobejecutar](#) en la AWS SDK for JavaScript API.

Escenarios

Comenzar a ejecutar rastreadores y trabajos

En el siguiente ejemplo de código, se muestra cómo:

- Crear un rastreador que rastree un bucket de Amazon S3 público y generar una base de datos de metadatos con formato CSV.
- Enumera información sobre bases de datos y tablas en tu AWS Glue Data Catalog.
- Crear un trabajo para extraer datos CSV del bucket de S3, transformar los datos y cargar el resultado con formato JSON en otro bucket de S3.
- Incluir información sobre las ejecuciones de trabajos, ver algunos de los datos transformados y limpiar los recursos.

Para obtener más información, consulte el [tutorial: Primeros pasos con AWS Glue Studio](#).

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree y ejecute un rastreador que rastree un bucket público de Amazon Simple Storage Service (Amazon S3) y genere una base de metadatos que describa los datos con formato CSV que encuentra.

```
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 };  
  };
```

Enumere información sobre bases de datos y tablas en su 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 };
};
```

Cree y ejecute un trabajo que extraiga datos CSV del bucket de Amazon S3 de origen, los transforme quitando campos y cambiándoles el nombre, y cargue el resultado con formato JSON en otro bucket de 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 };
  };
};
```

```
};
```

Incluya información sobre las ejecuciones de trabajos y vea algunos de los datos 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 };
  };
};
```

Elimine todos los recursos creados en la demostración.

```
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 {{ dbName: string[] }} */
      const { dbName } = 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 (dbName.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbName);
        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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobEjecute](#)
 - [GetJobCorre](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobCorre](#)

HealthImaging ejemplos que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con. HealthImaging

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola HealthImaging

Los siguientes ejemplos de código muestran cómo empezar a usarlo 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 obtener más información sobre la API, consulte [ListDatastores](#) la Referencia de AWS SDK for JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

CopyImageSet

En el siguiente ejemplo de código, se muestra cómo usar CopyImageSet.

SDK para JavaScript (v3)

Función de utilidad para copiar un conjunto de imágenes.

```
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 = "xxxxxxxxxxxx",
  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: 'xxxxxxxxxxxxxxxx',
  //   destinationImageSetProperties: {
  //     createdAt: 2023-09-27T19:46:21.824Z,
  //     imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxxx',
  //     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:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxxx',
```

```
//          imageSetState: 'LOCKED',  
//          imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',  
//          latestVersionId: '4',  
//          updatedAt: 2023-09-27T19:46:21.824Z  
//      }  
// }  
return response;  
};
```

Copiar un conjunto de imágenes sin destino.

```
try {  
  await copyImageSet(  
    "12345678901234567890123456789012",  
    "12345678901234567890123456789012",  
    "1"  
  );  
} catch (err) {  
  console.error(err);  
}
```

Copie un conjunto de imágenes con un destino.

```
try {  
  await copyImageSet(  
    "12345678901234567890123456789012",  
    "12345678901234567890123456789012",  
    "4",  
    "12345678901234567890123456789012",  
    "1"  
  );  
} catch (err) {  
  console.error(err);  
}
```

- Para obtener más información sobre la API, consulte [CopyImageSet](#) in AWS SDK for JavaScript API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

CreateDatastore

En el siguiente ejemplo de código, se muestra cómo 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- Para obtener más información sobre la API, consulte [CreateDatastore](#) la Referencia de AWS SDK for JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

DeleteDatastore

En el siguiente ejemplo de código, se muestra cómo 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- Para obtener más información sobre la API, consulte [DeleteDatastore](#) la Referencia de AWS SDK for JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

DeleteImageSet

En el siguiente ejemplo de código, se muestra cómo 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 = "xxxxxxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxxxxxx',
  //   imageSetState: 'LOCKED',
  //   imageSetWorkflowStatus: 'DELETING'
  // }
```

```
// }
return response;
};
```

- Para obtener más información sobre la API, consulte [DeleteImageSet](#) in AWS SDK for JavaScript API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

GetDICOMImportJob

En el siguiente ejemplo de código, se muestra cómo 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 = "xxxxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxxxx"
) => {
  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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//          endedAt: 2023-09-19T17:29:21.753Z,
//          inputS3Uri: 's3://healthimaging-source/CTStudy/',
//          jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//          jobName: 'job_1',
//          jobStatus: 'COMPLETED',
//          outputS3Uri: 's3://health-imaging-dest/
output_ct/'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'/',
//          submittedAt: 2023-09-19T17:27:25.143Z
//      }
// }

return response;
};

```

- Para obtener más información sobre la API, consulte [GetDICom ImportJob](#) en AWS SDK for JavaScript la referencia de la API.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

GetDatastore

En el siguiente ejemplo de código, se muestra cómo 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") => {

```


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 obtener información detallada sobre la API, consulte [GetImageFrame](#) in AWS SDK for JavaScript API Reference.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

GetImageSet

En el siguiente ejemplo de código, se muestra cómo 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 = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx",
  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,
```



```
/**
 * @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 = "xxxxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxxxx",
  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;
};
```

Obtener metadatos del conjunto de imágenes sin versión.

```
try {
  await getImageSetMetadata(
    "metadata.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012"
  );
} catch (err) {
  console.log("Error", err);
}
```

Obtener metadatos del conjunto de imágenes con la versión.

```
try {
  await getImageSetMetadata(
    "metadata2.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.log("Error", err);
}
```

- Para obtener más información sobre la API, consulte [GetImageSetMetadata](#) la Referencia de AWS SDK for JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

ListDICOMImportJobs

En el siguiente ejemplo de código, se muestra cómo 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 = "xxxxxxxxxxxxxxxxxxxxxx"
) => {
  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: 'xxxxxxxxxxxxxxxxxxxxxx',
  //       endedAt: 2023-09-22T14:49:51.351Z,
  //       jobId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //       jobName: 'test-1',
  //       jobStatus: 'COMPLETED',
  //       submittedAt: 2023-09-22T14:48:45.767Z
  //     }
  //   ]
  // }
```

```
// }
// ]}

return jobSummaries;
};
```

- Para obtener detalles sobre la API, consulte [ListDicom ImportJobs](#) en AWS SDK for JavaScript la referencia de la API.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

ListDatastores

En el siguiente ejemplo de código, se muestra cómo 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"]);
  }
}
```


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 = "xxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxx"
) => {
  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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//          imageSetState: 'ACTIVE',
//          versionId: '1'
//      }]
// }
return imageSetPropertiesList;
};
```

- Para obtener más información sobre la API, consulte [ListImageSetVersions](#) la Referencia de AWS SDK for JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

ListTagsForResource

En el siguiente ejemplo de código, se muestra cómo 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 obtener más información sobre la API, consulte [ListTagsForResource](#) la Referencia de AWS SDK for JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

SearchImageSets

En el siguiente ejemplo de código, se muestra cómo usar SearchImageSets.

SDK para JavaScript (v3)

La función de utilidad para buscar conjuntos de imágenes.

```
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 núm. 1: operador IGUAL.

```
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: el operador BETWEEN utiliza DICOM StudyDate y DICOM. StudyTime

```
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 núm. 3: el operador ENTRE usa CreatedAt. Los estudios de tiempo se habían mantenido previamente.

```
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 en el SeriesInstance UID de DICOM y BETWEEN en UpdatedAt y ordena la respuesta en orden ASC en el campo UpdatedAt.

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {updatedAt: new Date("1985-04-12T23:20:50.52Z")},
                ],
                operator: "BETWEEN",
            },
        ],
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

```
        {updatedAt: new Date()},
      ],
      operator: "BETWEEN",
    },
    {
      values: [
        {DICOMSeriesInstanceUID:
"1.1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
      ],
      operator: "EQUAL",
    },
  ],
  sort: {
    sortOrder: "ASC",
    sortField: "updatedAt",
  }
};

    await searchImageSets(datastoreId, searchCriteria);
  } catch (err) {
    console.error(err);
  }
}
```

- [Para obtener más información sobre la API, consulte Sets in API Reference. SearchImage AWS SDK for JavaScript](#)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

StartDICOMImportJob

En el siguiente ejemplo de código, se muestra cómo 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 obtener más información sobre la API, consulte [StartDicom ImportJob](#) en AWS SDK for JavaScript la referencia de la API.

Note

Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

TagResource

En el siguiente ejemplo de código, se muestra cómo 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 obtener más información sobre la API, consulte [TagResource](#) la Referencia de AWS SDK for JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

UntagResource

En el siguiente ejemplo de código, se muestra cómo 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 obtener más información sobre la API, consulte [UntagResource](#) la Referencia de AWS SDK for JavaScript la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

UpdateImageSetMetadata

En el siguiente ejemplo de código, se muestra cómo 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: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   imageSetState: 'LOCKED',
//   imageSetWorkflowStatus: 'UPDATING',
//   latestVersionId: '4',
//   updatedAt: 2023-09-27T19:41:43.494Z
// }
return response;
};

```

Caso de uso #1: insertar o actualizar un 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: eliminar un 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: eliminar una instancia.

```
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 obtener más información sobre la API, consulte [UpdateImageSetMetadata](#) la referencia AWS SDK for JavaScript de la API.

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Escenarios

Introducción a los conjuntos y marcos de imágenes

El siguiente ejemplo de código muestra cómo importar archivos DICOM y descargar marcos de imágenes en HealthImaging ellos.

La implementación está estructurada como una aplicación de línea de comandos de flujo de trabajo.

- Configure los recursos para una tarea de importación DICOM.
- Importe los archivos DICOM en un almacén de datos.
- Recupere los ID de los conjuntos de imágenes para el trabajo de importación.
- Recupere los ID de los marcos de imágenes para los conjuntos de imágenes.
- Descargue, decodifique y verifique los marcos de imágenes.
- Eliminación de recursos.

SDK para JavaScript (v3)

index.js- Organice los pasos.

```
// 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- Despliegue 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- Copia archivos 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- Comience a importar al almacén de datos.

```
// 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- Obtenga los ID de los conjuntos de imágenes.

```
// 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- Obtenga los ID de los marcos de imagen.

```

// 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- Verifica los marcos de las imágenes. Para la [verificación se utilizó la biblioteca de verificación de datos de AWS HealthImaging píxeles](#).

```

// 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 [seriesInstanceId, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceId, _] of Object.entries(series.Instances)) {
          console.log(
            `Verifying image set ${imageSetId} with series ${seriesInstanceId} and
sop ${sopInstanceId}`,
          );
          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- Destruye los 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 obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for JavaScript .
 - [DeleteImageSet](#)
 - [GetDicom ImportJob](#)
 - [GetImageMarco](#)
 - [GetImageSetMetadata](#)
 - [SearchImageConjuntos](#)
 - [StartDicom ImportJob](#)

Note

Hay más en marcha. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Etiquetar un almacén de datos

El siguiente ejemplo de código muestra cómo etiquetar un banco HealthImaging de datos.

SDK para JavaScript (v3)

Para etiquetar un almacén de datos

```
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);
}
```

Función de utilidad para etiquetar un 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;
};

```

Para enumerar las etiquetas de almacenes de datos

```

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);
}

```

La función de utilidad para enumerar las etiquetas de un 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;
  };

```

Para desetiquetar un almacén de datos

```

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);
}

```

La función de utilidad para eliminar la etiqueta de un 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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Etiquetar un conjunto de imágenes

El siguiente ejemplo de código muestra cómo etiquetar un conjunto HealthImaging de imágenes.

SDK para JavaScript (v3)

Para etiquetar un conjunto de imágenes

```
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);
}
```

Función de utilidad para etiquetar un 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;  
};
```

Para enumerar las etiquetas de un conjunto de imágenes

```
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);  
}
```

La función de utilidad para enumerar las etiquetas de un 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;
};

```

Para desetiquetar un conjunto de imágenes

```

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);
}

```

La función de utilidad para eliminar la etiqueta de un 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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejemplos de IAM que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la AWS SDK for JavaScript (v3) con IAM.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a IAM

En los siguientes ejemplos de código se muestra cómo empezar a utilizar IAM.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [ListPolicies](#) la Referencia AWS SDK for JavaScript de la API.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

AttachRolePolicy

En el siguiente ejemplo de código, se muestra cómo usar `AttachRolePolicy`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Asocie la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [AttachRolela política](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [AttachRolela política](#) en la referencia AWS SDK for JavaScript de la API.

CreateAccessKey

En el siguiente ejemplo de código, se muestra cómo usar CreateAccessKey.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree la clave de acceso.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateAccessKey](#) la referencia sobre la clave [AWS SDK for JavaScript](#) de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).

- Para obtener más información sobre la API, consulta [CreateAccessla referencia sobre la clave AWS SDK for JavaScript](#) de la API.

CreateAccountAlias

En el siguiente ejemplo de código, se muestra cómo usar CreateAccountAlias.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el alias de la cuenta.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateAccountAlias](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateAccountAlias](#) en la referencia AWS SDK for JavaScript de la API.

CreateGroup

En el siguiente ejemplo de código, se muestra cómo usar CreateGroup.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateGroup](#) la Referencia AWS SDK for JavaScript de la API.

CreateInstanceProfile

En el siguiente ejemplo de código, se muestra cómo usar CreateInstanceProfile.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```


- Para obtener más información sobre la API, consulta el [CreateInstanceperfil](#) en la referencia AWS SDK for JavaScript de la API.

CreatePolicy

En el siguiente ejemplo de código, se muestra cómo usar CreatePolicy.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreatePolicy](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreatePolicy](#) la Referencia AWS SDK for JavaScript de la API.

CreateRole

En el siguiente ejemplo de código, se muestra cómo usar CreateRole.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el rol.

```
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 obtener más información sobre la API, consulta [CreateRole](#) la Referencia AWS SDK for JavaScript de la API.

CreateSAMLProvider

En el siguiente ejemplo de código, se muestra cómo usar CreateSAMLProvider.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener información sobre la API, consulte [CreateSAMLProvider](#) en la Referencia de la API de AWS SDK for JavaScript .

CreateServiceLinkedRole

En el siguiente ejemplo de código, se muestra cómo usar CreateServiceLinkedRole.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree un rol vinculado al servicio.

```
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 obtener más información sobre la API, consulta [CreateServiceLinkedRole](#) la Referencia AWS SDK for JavaScript de la API.

CreateUser

En el siguiente ejemplo de código, se muestra cómo usar CreateUser.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el usuario.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateUser](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateUser](#) la Referencia AWS SDK for JavaScript de la API.

DeleteAccessKey

En el siguiente ejemplo de código, se muestra cómo usar DeleteAccessKey.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine la clave de acceso.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).

- Para obtener más información sobre la API, consulta [DeleteAccessla referencia sobre la clave AWS SDK for JavaScript](#) de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteAccessla referencia sobre la clave AWS SDK for JavaScript](#) de la API.

DeleteAccountAlias

En el siguiente ejemplo de código, se muestra cómo usar DeleteAccountAlias.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine el alias de cuenta.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteAccountAlias](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteAccountAlias](#) en la referencia AWS SDK for JavaScript de la API.

DeleteGroup

En el siguiente ejemplo de código, se muestra cómo usar DeleteGroup.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteGroup](#) la Referencia AWS SDK for JavaScript de la API.

DeleteInstanceProfile

En el siguiente ejemplo de código, se muestra cómo usar DeleteInstanceProfile.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const client = new IAMClient({});  
await client.send(  
  new DeleteInstanceProfileCommand({  
    InstanceProfileName: NAMES.instanceProfileName,  
  }),  
);
```

- Para obtener más información sobre la API, consulta el [DeleteInstanceperfil](#) en la referencia AWS SDK for JavaScript de la API.

DeletePolicy

En el siguiente ejemplo de código, se muestra cómo usar DeletePolicy.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine la 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 obtener más información sobre la API, consulta [DeletePolicy](#) la Referencia AWS SDK for JavaScript de la API.

DeleteRole

En el siguiente ejemplo de código, se muestra cómo usar DeleteRole.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine el rol.

```
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 obtener más información sobre la API, consulta [DeleteRole](#) la Referencia AWS SDK for JavaScript de la API.

DeleteRolePolicy

En el siguiente ejemplo de código, se muestra cómo usar DeleteRolePolicy.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteRolela política](#) en la referencia AWS SDK for JavaScript de la API.

DeleteSAMLProvider

En el siguiente ejemplo de código, se muestra cómo usar DeleteSAMLProvider.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener información sobre la API, consulte [DeleteSAMLProvider](#) en la Referencia de la API de AWS SDK for JavaScript .

DeleteServerCertificate

En el siguiente ejemplo de código, se muestra cómo usar DeleteServerCertificate.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine un 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta el [DeleteServercertificado](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta el [DeleteServercertificado](#) en la referencia AWS SDK for JavaScript de la API.

DeleteServiceLinkedRole

En el siguiente ejemplo de código, se muestra cómo usar DeleteServiceLinkedRole.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteServiceLinkedRole](#) la Referencia AWS SDK for JavaScript de la API.

DeleteUser

En el siguiente ejemplo de código, se muestra cómo usar DeleteUser.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine el usuario.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteUser](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteUser](#) la Referencia AWS SDK for JavaScript de la API.

DetachRolePolicy

En el siguiente ejemplo de código, se muestra cómo usar DetachRolePolicy.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Desasocie la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DetachRole política](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DetachRolela política](#) en la referencia AWS SDK for JavaScript de la API.

GetAccessKeyLastUsed

En el siguiente ejemplo de código, se muestra cómo usar GetAccessKeyLastUsed.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga una clave de acceso.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta Cómo [GetAccessKeyLastse usa](#) en la referencia de la AWS SDK for JavaScript API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta Cómo [GetAccessKeyLastse usa](#) en la referencia de la AWS SDK for JavaScript API.

GetAccountPasswordPolicy

En el siguiente ejemplo de código, se muestra cómo usar `GetAccountPasswordPolicy`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga la política de contraseñas de la cuenta.

```
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 obtener más información sobre la API, consulta [GetAccountPasswordPolicy](#) la Referencia AWS SDK for JavaScript de la API.

GetPolicy

En el siguiente ejemplo de código, se muestra cómo usar GetPolicy.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetPolicy](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetPolicy](#) la Referencia AWS SDK for JavaScript de la API.

GetRole

En el siguiente ejemplo de código, se muestra cómo usar `GetRole`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga el rol.

```
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 obtener más información sobre la API, consulta [GetRole](#) la Referencia AWS SDK for JavaScript de la API.

GetServerCertificate

En el siguiente ejemplo de código, se muestra cómo usar `GetServerCertificate`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga un certificado de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta el [GetServercertificado](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta el [GetServercertificado](#) en la referencia AWS SDK for JavaScript de la API.

GetServiceLinkedRoleDeletionStatus

En el siguiente ejemplo de código, se muestra cómo usar `GetServiceLinkedRoleDeletionStatus`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [GetServiceLinkedRoleDeletionStatus](#) la Referencia AWS SDK for JavaScript de la API.

ListAccessKeys

En el siguiente ejemplo de código, se muestra cómo usar `ListAccessKeys`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere las claves de acceso.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [ListAccessección Claves](#) en la referencia de la AWS SDK for JavaScript API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [ListAccessection Claves](#) en la referencia de la AWS SDK for JavaScript API.

ListAccountAliases

En el siguiente ejemplo de código, se muestra cómo usar ListAccountAliases.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los alias de cuenta.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [ListAccountsección Alias](#) en la referencia de AWS SDK for JavaScript la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [ListAccountsección Alias](#) en la referencia de AWS SDK for JavaScript la API.

ListAttachedRolePolicies

En el siguiente ejemplo de código, se muestra cómo usar ListAttachedRolePolicies.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere las políticas que están asociadas a un rol.

```
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 obtener más información sobre la API, consulta [ListAttachedRolePolicies](#) la Referencia AWS SDK for JavaScript de la API.

ListGroups

En el siguiente ejemplo de código, se muestra cómo usar ListGroups.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los 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 obtener más información sobre la API, consulta [ListGroups](#) la Referencia AWS SDK for JavaScript de la API.

ListPolicies

En el siguiente ejemplo de código, se muestra cómo usar `ListPolicies`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere las 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 obtener más información sobre la API, consulta [ListPolicies](#) la Referencia AWS SDK for JavaScript de la API.

ListRolePolicies

En el siguiente ejemplo de código, se muestra cómo usar `ListRolePolicies`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere las 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 obtener más información sobre la API, consulta [ListRolelas políticas](#) en la referencia AWS SDK for JavaScript de la API.

ListRoles

En el siguiente ejemplo de código, se muestra cómo usar ListRoles.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los roles.

```
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 obtener más información sobre la API, consulta [ListRoles](#) la Referencia AWS SDK for JavaScript de la API.

ListSAMLProviders

En el siguiente ejemplo de código, se muestra cómo usar `ListSAMLProviders`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los proveedores de 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 obtener información sobre la API, consulte [ListSAMLProviders](#) en la Referencia de la API de AWS SDK for JavaScript .

ListServerCertificates

En el siguiente ejemplo de código, se muestra cómo usar `ListServerCertificates`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [ListServersección Certificados](#) en la referencia de la AWS SDK for JavaScript API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [ListServersección Certificados](#) en la referencia de la AWS SDK for JavaScript API.

ListUsers

En el siguiente ejemplo de código, se muestra cómo usar ListUsers.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere los usuarios.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListUsers](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListUsers](#) la Referencia AWS SDK for JavaScript de la API.

PutRolePolicy

En el siguiente ejemplo de código, se muestra cómo usar PutRolePolicy.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [PutRolela política](#) en la referencia AWS SDK for JavaScript de la API.

UpdateAccessKey

En el siguiente ejemplo de código, se muestra cómo usar UpdateAccessKey.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Actualice la clave de acceso.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [UpdateAccessla referencia sobre la clave AWS SDK for JavaScript](#) de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [UpdateAccessla referencia sobre la clave AWS SDK for JavaScript](#) de la API.

UpdateServerCertificate

En el siguiente ejemplo de código, se muestra cómo usar UpdateServerCertificate.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Actualice un certificado de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta el [UpdateServercertificado](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta el [UpdateServercertificado](#) en la referencia AWS SDK for JavaScript de la API.

UpdateUser

En el siguiente ejemplo de código, se muestra cómo usar UpdateUser.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Actualice el usuario.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [UpdateUser](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [UpdateUser](#) la Referencia AWS SDK for JavaScript de la API.

UploadServerCertificate

En el siguiente ejemplo de código, se muestra cómo usar UploadServerCertificate.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta el [UploadServercertificado](#) en la referencia AWS SDK for JavaScript de la API.

Escenarios

Creación y administración de un servicio resiliente

El siguiente ejemplo de código muestra cómo crear un servicio web con equilibrio de carga que muestre recomendaciones de libros, películas y canciones. El ejemplo muestra cómo responde el servicio a los errores y cómo reestructurarlo para aumentar la resiliencia cuando se produzcan errores.

- Utilice un grupo de Amazon EC2 Auto Scaling para crear instancias de Amazon Elastic Compute Cloud (Amazon EC2) basadas en una plantilla de lanzamiento y para mantener el número de instancias dentro de un rango específico.
- Administre y distribuya las solicitudes HTTP con Elastic Load Balancing.
- Supervise el estado de las instancias de un grupo de escalado automático y reenvíe las solicitudes solo a las instancias en buen estado.
- Ejecute un servidor web Python en cada instancia de EC2 para administrar las solicitudes HTTP. El servidor web responde con recomendaciones y comprobaciones de estado.
- Simule un servicio de recomendaciones con una tabla de Amazon DynamoDB.
- Controle la respuesta del servidor web a las solicitudes y las comprobaciones de estado actualizando AWS Systems Manager los parámetros.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecute el escenario interactivo en un símbolo del sistema.

```
#!/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);
}
```

Cree los pasos para implementar todos los 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,
    );
  }
}),
];
```

Cree los pasos para ejecutar la demostración.

```
// 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;
}
```

Cree los pasos para destruir todos los 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 obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for JavaScript .
 - [AttachLoadBalancerTargetGrupos](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstancePerfil](#)
 - [CreateLaunchPlantilla](#)
 - [CreateListener](#)
 - [CreateLoadEquilibrador](#)
 - [CreateTargetGrupo](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstancePerfil](#)
 - [DeleteLaunchPlantilla](#)
 - [DeleteLoadEquilibrador](#)
 - [DeleteTargetGrupo](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZonas](#)
 - [DescribeIamInstanceProfileAsociaciones](#)
 - [DescribeInstances](#)
 - [DescribeLoadEquilibradores](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGrupos](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)
 - [ReplacelamInstanceProfileAsociación](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

Crear un usuario y asumir un rol

En el siguiente ejemplo de código, se muestra cómo crear un usuario y asumir un rol.

Warning

Para evitar riesgos de seguridad, no utilice a los usuarios de IAM para la autenticación cuando desarrolle software especialmente diseñado o trabaje con datos reales. En cambio, utilice la federación con un proveedor de identidades como [AWS IAM Identity Center](#).

- Crear un usuario que no tenga permisos.
- Crear un rol que conceda permiso para enumerar los buckets de Amazon S3 para la cuenta.
- Agregar una política para que el usuario asuma el rol.
- Asumir el rol y enumerar los buckets de S3 con credenciales temporales, y después limpiar los recursos.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree un usuario de IAM y un rol que conceda permiso para enumerar los buckets de Amazon S3. El usuario solo tiene derechos para asumir el rol. Después de asumir el rol, use las credenciales temporales para enumerar los buckets de la cuenta.

```
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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [AttachRolePolítica](#)
 - [CreateAccessClave](#)
 - [CreatePolicy](#)
 - [CreateRole](#)

- [CreateUser](#)
- [DeleteAccessClave](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolítica](#)
- [DetachRolePolítica](#)
- [PutUserPolítica](#)

Ejemplos de Kinesis con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la AWS SDK for JavaScript (v3) con Kinesis.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas


- [Ejemplos sin servidor](#)

Ejemplos sin servidor

Invocar una función de Lambda desde un desencadenador de Kinesis

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al recibir registros de un flujo de Kinesis. La función recupera la carga útil de Kinesis, la decodifica desde Base64 y registra el contenido del registro.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de Kinesis con Lambda mediante. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Consumir un evento de Kinesis con Lambda mediante. 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;
}
```

Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Kinesis

En el siguiente ejemplo de código se muestra cómo implementar una respuesta por lotes parcial para funciones de Lambda que reciben eventos de un flujo de Kinesis. La función informa los errores de

los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante 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;
}
```

Cómo informar de errores en los artículos de lote de Kinesis con Lambda mediante 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;
  }
}
```

Ejemplos de Lambda con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Lambda.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Lambda

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Lambda.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Para obtener más información sobre la API, consulta [ListFunctions](#) la Referencia AWS SDK for JavaScript de la API.

Temas

- [Acciones](#)
- [Escenarios](#)
- [Ejemplos sin servidor](#)

Acciones

CreateFunction

En el siguiente ejemplo de código, se muestra cómo usar CreateFunction.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [CreateFunction](#) la Referencia AWS SDK for JavaScript de la API.

DeleteFunction

En el siguiente ejemplo de código, se muestra cómo usar DeleteFunction.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [DeleteFunction](#) la Referencia AWS SDK for JavaScript de la API.

GetFunction

En el siguiente ejemplo de código, se muestra cómo usar `GetFunction`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getFunction = (funcName) => {  
  const client = new LambdaClient({});  
  const command = new GetFunctionCommand({ FunctionName: funcName });  
  return client.send(command);  
};
```

- Para obtener más información sobre la API, consulta [GetFunction](#) la Referencia AWS SDK for JavaScript de la API.

Invoke

En el siguiente ejemplo de código, se muestra cómo usar `Invoke`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).


```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- Para obtener información sobre la API, consulte [Invocar](#) en la referencia de la API de AWS SDK for JavaScript .

ListFunctions

En el siguiente ejemplo de código, se muestra cómo usar `ListFunctions`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [ListFunctions](#) la Referencia AWS SDK for JavaScript de la API.

UpdateFunctionCode

En el siguiente ejemplo de código, se muestra cómo usar `UpdateFunctionCode`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta el [UpdateFunctioncódigo](#) en la referencia AWS SDK for JavaScript de la API.

UpdateFunctionConfiguration

En el siguiente ejemplo de código, se muestra cómo usar `UpdateFunctionConfiguration`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}./functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

- Para obtener más información sobre la API, consulta [UpdateFunctionla sección Configuración](#) en la referencia de la AWS SDK for JavaScript API.

Escenarios

Comenzar a usar las funciones

En el siguiente ejemplo de código, se muestra cómo:

- Crear un rol de IAM y una función de Lambda y, a continuación, cargar el código de controlador.
- Invocar la función con un único parámetro y obtener resultados.
- Actualizar el código de la función y configurar con una variable de entorno.
- Invocar la función con un nuevo parámetro y obtener resultados. Mostrar el registro de ejecución devuelto.
- Enumerar las funciones de su cuenta y, luego, limpiar los recursos.

Para obtener información, consulte [Crear una función de Lambda con la consola](#).

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree un rol AWS Identity and Access Management (de IAM) que conceda permiso a Lambda para escribir en los 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);
};
```

Cree una función de Lambda y cargue el código de controlador.

```
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 la función con un único parámetro y obtenga 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 };
};
```

Actualice el código de la función y configure su entorno Lambda con una variable de entorno.

```
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);
};

```

Enumere las funciones de su cuenta.

```

const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};

```

Elimine el rol de IAM y la función de 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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Invoke](#)
 - [ListFunctions](#)
 - [UpdateFunctionCódigo](#)
 - [UpdateFunctionConfiguración](#)

Ejemplos sin servidor

Invocar una función de Lambda desde un desencadenador de Kinesis

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al recibir registros de un flujo de Kinesis. La función recupera la carga útil de Kinesis, la decodifica desde Base64 y registra el contenido del registro.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de Kinesis con Lambda mediante. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
```

```

    console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
    const recordData = await getRecordDataAsync(record.kinesis);
    console.log(`Record Data: ${recordData}`);
    // TODO: Do interesting work based on the new data
  } catch (err) {
    console.error(`An error occurred ${err}`);
    throw err;
  }
}
console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

Consumir un evento de Kinesis con Lambda mediante 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;
}

```

Invocación de una función de Lambda desde un desencadenador de DynamoDB

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento que se desencadena al recibir registros de una transmisión de DynamoDB. La función recupera la carga útil de DynamoDB y registra el contenido del registro.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de DynamoDB con Lambda mediante JavaScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
}

```

```
    event.Records.forEach(record => {
      logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consumir un evento de DynamoDB con Lambda mediante 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 una función Lambda desde un activador de Amazon DocumentDB

El siguiente ejemplo de código muestra cómo implementar una función de Lambda que recibe un evento que se desencadena al recibir registros de un flujo de cambios de DocumentDB. La función recupera la carga útil de DocumentDB y registra el contenido del registro.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de Amazon DocumentDB con Lambda mediante. 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));
};
```

Invocación de una función de Lambda desde un desencadenador de Amazon S3

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al cargar un objeto en un bucket de S3. La función recupera el nombre del bucket de S3 y la clave del objeto del parámetro de evento y llama a la API de Amazon S3 para recuperar y registrar el tipo de contenido del objeto.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de S3 con Lambda mediante. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";
```

```

const client = new S3Client();

exports.handler = async (event, context) => {

  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
  '));

  try {
    const { ContentType } = await client.send(new HeadObjectCommand({
      Bucket: bucket,
      Key: key,
    }));

    console.log('CONTENT TYPE:', ContentType);
    return ContentType;

  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make
    sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};

```

Consumir un evento de S3 con Lambda mediante TypeScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
  '));
  const params = {

```

```
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

Invocar una función de Lambda desde un desencadenador de Amazon SNS

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al recibir mensajes de un tema de SNS. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

SDK para JavaScript (v3)

Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de SNS con JavaScript Lambda mediante.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};
```

```
async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumir un evento de SNS con TypeScript Lambda mediante.

```
// 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 una función de Lambda desde un desencadenador de Amazon SQS

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al recibir mensajes de una cola de SQS. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de SQS con JavaScript Lambda mediante.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumir un evento de SQS con TypeScript Lambda mediante.

```
// 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;
  }
}
```

Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Kinesis

En el siguiente ejemplo de código se muestra cómo implementar una respuesta por lotes parcial para funciones de Lambda que reciben eventos de un flujo de Kinesis. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante 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;
}
```

Cómo informar de errores en los artículos de lote de Kinesis con Lambda mediante 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;
}
```

Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB

El siguiente ejemplo de código muestra cómo implementar una respuesta por lotes parcial para las funciones de Lambda que reciben eventos de una transmisión de DynamoDB. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

SDK para (v3 JavaScript)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Cómo informar de errores en los elementos de lote de DynamoDB con Lambda mediante 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: [] };
};
```

Cómo informar de errores en los elementos de lote de DynamoDB con Lambda mediante 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
}
```

Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS.

En el siguiente ejemplo de código se muestra cómo implementar una respuesta por lotes parcial para funciones de Lambda que reciben eventos de una cola de SQS. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

SDK para (v3 JavaScript)

 Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Informe de errores de artículos de lotes de SQS con JavaScript Lambda mediante.

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

Informe de errores de artículos de lotes de SQS con TypeScript Lambda mediante.

```
// 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.');
```

Ejemplos de Amazon Personalize con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Personalize.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

CreateBatchInferenceJob

En el siguiente ejemplo de código, se muestra cómo usar CreateBatchInferenceJob.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateBatchInferenceJob](#) la Referencia AWS SDK for JavaScript de la API.

CreateBatchSegmentJob

En el siguiente ejemplo de código, se muestra cómo usar `CreateBatchSegmentJob`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateBatchSegmentJob](#) la Referencia AWS SDK for JavaScript de la API.

CreateCampaign

En el siguiente ejemplo de código, se muestra cómo usar CreateCampaign.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateCampaign](#) la Referencia AWS SDK for JavaScript de la API.

CreateDataset

En el siguiente ejemplo de código, se muestra cómo usar CreateDataset.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateDataset](#) la Referencia AWS SDK for JavaScript de la API.

CreateDatasetExportJob

En el siguiente ejemplo de código, se muestra cómo usar CreateDatasetExportJob.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateDatasetExportJob](#) la Referencia AWS SDK for JavaScript de la API.

CreateDatasetGroup

En el siguiente ejemplo de código, se muestra cómo usar CreateDatasetGroup.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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);
```

Crear un grupo de conjuntos de datos de dominios.

```
// 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 obtener más información sobre la API, consulta [CreateDatasetAgrupar](#) en la referencia de la AWS SDK for JavaScript API.

CreateDatasetImportJob

En el siguiente ejemplo de código, se muestra cómo usar CreateDatasetImportJob.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateDatasetImportJob](#) la Referencia AWS SDK for JavaScript de la API.

CreateEventTracker

En el siguiente ejemplo de código, se muestra cómo usar CreateEventTracker.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateEventTracker](#) en la referencia AWS SDK for JavaScript de la API.

CreateFilter

En el siguiente ejemplo de código, se muestra cómo usar CreateFilter.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateFilter](#) la Referencia AWS SDK for JavaScript de la API.

CreateRecommender

En el siguiente ejemplo de código, se muestra cómo usar CreateRecommender.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateRecommender](#) la Referencia AWS SDK for JavaScript de la API.

CreateSchema

En el siguiente ejemplo de código, se muestra cómo usar CreateSchema.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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();
```

Crea un esquema con un dominio.

```
// 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 obtener más información sobre la API, consulte [CreateSchema](#) la referencia AWS SDK for JavaScript de la API.

CreateSolution

En el siguiente ejemplo de código, se muestra cómo usar CreateSolution.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateSolution](#) la Referencia AWS SDK for JavaScript de la API.

CreateSolutionVersion

En el siguiente ejemplo de código, se muestra cómo usar `CreateSolutionVersion`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [CreateSolutionla versión](#) en la referencia AWS SDK for JavaScript de la API.

Ejemplos de Amazon Personalize Events con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Personalize Events.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

PutEvents

En el siguiente ejemplo de código, se muestra cómo usar PutEvents.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [PutEvents](#) la Referencia AWS SDK for JavaScript de la API.

PutItems

En el siguiente ejemplo de código, se muestra cómo usar PutItems.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [PutItems](#) la Referencia AWS SDK for JavaScript de la API.

PutUsers

En el siguiente ejemplo de código, se muestra cómo usar PutUsers.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [PutUsers](#) la Referencia AWS SDK for JavaScript de la API.

Ejemplos de Amazon Personalize Runtime mediante el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Personalize Runtime.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

GetPersonalizedRanking

En el siguiente ejemplo de código, se muestra cómo usar `GetPersonalizedRanking`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre las API, consulta la [GetPersonalizedsección Clasificación](#) en la referencia de las AWS SDK for JavaScript API.

GetRecommendations

En el siguiente ejemplo de código, se muestra cómo usar `GetRecommendations`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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();
```

Obtener recomendaciones con un filtro (grupo de conjuntos de datos 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();
```

Obtenga recomendaciones filtradas de un recomendador creado en un grupo de conjuntos de datos de dominio.

```
// 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 obtener más información sobre la API, consulta [GetRecommendations](#) la Referencia AWS SDK for JavaScript de la API.

Ejemplos de Amazon Pinpoint con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Pinpoint.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

SendMessage

En el siguiente ejemplo de código, se muestra cómo usar SendMessage.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
export const pinClient = new PinpointClient({ region: REGION });
```

Enviar un mensaje de correo electrónico.

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessageCommand } 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 = `
```

```
<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();

```

Enviar un mensaje 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., XXXXXXXX66e4e9986478cXXXXXXXXXX

/* 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 obtener más información sobre la API, consulta [SendMessages](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enviar un mensaje de correo electrónico.

```
"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 = `
<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"]
    );
  }
});
```

Enviar un mensaje 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.sendMessage(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 obtener más información sobre la API, consulta [SendMessages](#) la Referencia AWS SDK for JavaScript de la API.

Ejemplos de Amazon Redshift con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Redshift.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

CreateCluster

En el siguiente ejemplo de código, se muestra cómo usar CreateCluster.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 };
```

Cree el clúster.

```
// 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 obtener más información sobre la API, consulta [CreateCluster](#) la Referencia AWS SDK for JavaScript de la API.

DeleteCluster

En el siguiente ejemplo de código, se muestra cómo usar DeleteCluster.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 };
```

Cree el clúster.

```
// 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 obtener más información sobre la API, consulta [DeleteCluster](#) la Referencia AWS SDK for JavaScript de la API.

DescribeClusters

En el siguiente ejemplo de código, se muestra cómo usar DescribeClusters.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 };
```

Describir sus clústeres.

```
// 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 obtener más información sobre la API, consulta [DescribeClusters](#) la Referencia AWS SDK for JavaScript de la API.

ModifyCluster

En el siguiente ejemplo de código, se muestra cómo usar ModifyCluster.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 un clúster.

```
// 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 obtener más información sobre la API, consulta [ModifyCluster](#) la Referencia AWS SDK for JavaScript de la API.

Ejemplos de Amazon S3 que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon S3.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon S3

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon S3.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [ListBuckets](#) la Referencia AWS SDK for JavaScript de la API.

Temas

- [Acciones](#)
- [Escenarios](#)
- [Ejemplos sin servidor](#)

Acciones

CopyObject

En el siguiente ejemplo de código, se muestra cómo usar CopyObject.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Copie el 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 obtener más información sobre la API, consulta [CopyObject](#) la Referencia AWS SDK for JavaScript de la API.

CreateBucket

En el siguiente ejemplo de código, se muestra cómo usar CreateBucket.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Crear el 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateBucket](#) la Referencia AWS SDK for JavaScript de la API.

DeleteBucket

En el siguiente ejemplo de código, se muestra cómo usar DeleteBucket.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine el 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteBucket](#) la Referencia AWS SDK for JavaScript de la API.

DeleteBucketPolicy

En el siguiente ejemplo de código, se muestra cómo usar DeleteBucketPolicy.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine la política del 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteBucketla política](#) en la referencia AWS SDK for JavaScript de la API.

DeleteBucketWebsite

En el siguiente ejemplo de código, se muestra cómo usar DeleteBucketWebsite.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine la configuración del sitio web del 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta el [DeleteBucket sitio web](#) en la sección de referencia de la AWS SDK for JavaScript API.

DeleteObject

En el siguiente ejemplo de código, se muestra cómo usar DeleteObject.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine un 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 obtener más información sobre la API, consulta [DeleteObject](#) la Referencia AWS SDK for JavaScript de la API.

DeleteObjects

En el siguiente ejemplo de código, se muestra cómo usar DeleteObjects.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Elimine varios 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 obtener más información sobre la API, consulta [DeleteObjects](#) la Referencia AWS SDK for JavaScript de la API.

GetBucketAcl

En el siguiente ejemplo de código, se muestra cómo usar `GetBucketAcl`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga los permisos 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulte [GetBucketAcl](#) en la referencia AWS SDK for JavaScript de la API.

GetBucketCors

En el siguiente ejemplo de código, se muestra cómo usar GetBucketCors.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga la política de CORS para el 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulte [GetBucketCors](#) en la referencia de AWS SDK for JavaScript la API.

GetBucketPolicy

En el siguiente ejemplo de código, se muestra cómo usar `GetBucketPolicy`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga la política del 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetBucketla política](#) en la referencia AWS SDK for JavaScript de la API.

GetBucketWebsite

En el siguiente ejemplo de código, se muestra cómo usar `GetBucketWebsite`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtenga la configuración de sitio web.

```
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 obtener más información sobre la API, consulta el [GetBucket sitio web](#) en la sección de referencia de la AWS SDK for JavaScript API.

GetObject

En el siguiente ejemplo de código, se muestra cómo usar GetObject.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Descargue el 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetObject](#) la Referencia AWS SDK for JavaScript de la API.

GetObjectLockConfiguration

En el siguiente ejemplo de código, se muestra cómo usar `GetObjectLockConfiguration`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [GetObjectLockConfiguration](#) la Referencia AWS SDK for JavaScript de la API.

GetObjectRetention

En el siguiente ejemplo de código, se muestra cómo usar `GetObjectRetention`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta la sección [GetObjectRetención](#) en la referencia de la AWS SDK for JavaScript API.

ListBuckets

En el siguiente ejemplo de código, se muestra cómo usar `ListBuckets`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtener una lista de los 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListBuckets](#) la Referencia AWS SDK for JavaScript de la API.

ListObjectsV2

En el siguiente ejemplo de código, se muestra cómo usar ListObjectsV2.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumere todos los objetos del bucket. Si hay más de un objeto, IsTruncated se NextContinuationToken usará para recorrer toda la lista.

```
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 obtener más información sobre la API, consulta la [ListObjectsversión 2](#) en la referencia AWS SDK for JavaScript de la API.

PutBucketAcl

En el siguiente ejemplo de código, se muestra cómo usar PutBucketAcl.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Coloque la ACL del 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulte [PutBucketAcl](#) en la referencia AWS SDK for JavaScript de la API.

PutBucketCors

En el siguiente ejemplo de código, se muestra cómo usar PutBucketCors.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Añada una regla 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulte [PutBucketCors](#) en la referencia de AWS SDK for JavaScript la API.

PutBucketPolicy

En el siguiente ejemplo de código, se muestra cómo usar PutBucketPolicy.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Añada la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutBucketla política](#) en la referencia AWS SDK for JavaScript de la API.

PutBucketWebsite

En el siguiente ejemplo de código, se muestra cómo usar PutBucketWebsite.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Establezca la configuración de sitio web.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta el [PutBucketWebsite](#) en la sección de referencia de la AWS SDK for JavaScript API.

PutObject

En el siguiente ejemplo de código, se muestra cómo usar PutObject.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cargue el 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [PutObject](#) la Referencia AWS SDK for JavaScript de la API.

PutObjectLegalHold

En el siguiente ejemplo de código, se muestra cómo usar PutObjectLegalHold.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [PutObjectLegalHold](#) la Referencia AWS SDK for JavaScript de la API.

PutObjectLockConfiguration

En el siguiente ejemplo de código, se muestra cómo usar PutObjectLockConfiguration.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Establecimiento de la configuración de bloqueo de objetos de un 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");
}

```

Establecimiento del período de retención predeterminado de un 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 obtener más información sobre la API, consulta [PutObjectLockConfiguration](#) la Referencia AWS SDK for JavaScript de la API.

PutObjectRetention

En el siguiente ejemplo de código, se muestra cómo usar PutObjectRetention.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta la sección [PutObjectRetención](#) en la referencia de la AWS SDK for JavaScript API.

Escenarios

Crear una URL prefirmada

En el siguiente ejemplo de código se muestra cómo crear una URL prefirmada para Amazon S3 y cargar un objeto.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree una URL prefirmada para cargar un objeto en un 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);
}
};
```

Cree una URL prefirmada para descargar un objeto de un 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).

Crear una página web que enumere los objetos de Amazon S3

En los siguientes ejemplos de código se muestra cómo obtener una lista de los objetos de Amazon S3 en una página web.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

El siguiente código es el componente de React relevante que realiza llamadas al AWS SDK. Puede encontrar una versión ejecutable de la aplicación que contiene este componente en el enlace 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 obtener más información sobre la API, consulte la Referencia de [ListObjects](#) la AWS SDK for JavaScript API.

Comenzar a usar buckets y objetos

En el siguiente ejemplo de código, se muestra cómo:

- Creación de un bucket y cargar un archivo en el bucket.
- Descargar un objeto desde un bucket.
- Copiar un objeto en una subcarpeta de un bucket.
- Obtención de una lista de los objetos de un bucket.
- Eliminación del bucket y todos los objetos que incluye.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Primero, importe todos los módulos necesarios.

```
// 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";

```

Las importaciones anteriores hacen referencia a algunas utilidades auxiliares. Estas utilidades son locales del GitHub repositorio vinculado al principio de esta sección. Consulte las siguientes implementaciones de esas utilidades a modo de referencia.

```

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`;
};

```

Los objetos de S3 se almacenan en buckets (cubos). Definamos una función para crear un nuevo 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;
};

```

Los buckets contienen «objetos». Esta función carga el contenido de un directorio al bucket en forma de 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.`);
}
};
```

Después de cargar los objetos, confirme que se hayan subido correctamente. Puede utilizarlas `ListObjects` para eso. Utilizará la propiedad «Clave», pero también hay otras propiedades útiles en la respuesta.

```
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");
};
```

A veces, es posible que quiera copiar un objeto de un bucket en otro bucket. Usa el `CopyObject` comando para eso.

```
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();
}
```

No existe ningún método de SDK para obtener varios objetos de un bucket. En su lugar, creará una lista de objetos para descargarlos e iterarlos.

```
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");
};

```

Ha llegado el momento de limpiar los recursos. Un bucket debe estar vacío para poder eliminarlo. Estas dos funciones vacían y eliminan el 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`);
};

```

La función «principal» reúne todo. Si ejecuta este archivo directamente, se llamará a la función principal.

```

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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)

- [DeleteObjects](#)
- [GetObject](#)
- [ListObjectsV2](#)
- [PutObject](#)

Obtención de la configuración de retención legal de un objeto

En el siguiente ejemplo de código, se muestra cómo obtener la configuración de retención legal de un bucket de S3.

SDK para JavaScript (v3)

 Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [GetObjectLegalHold](#) la Referencia AWS SDK for JavaScript de la API.

Bloqueo de objetos de Amazon S3

En el siguiente ejemplo de código, se muestra cómo trabajar con características de bloqueo de objetos de S3.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

index.js- Punto de entrada al flujo de trabajo. Esto organiza todos los pasos. Visite GitHub para ver los detalles de implementación de Scenario ScenarioInput, ScenarioOutput, y ScenarioAction.

```
// 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- Envía mensajes de bienvenida a la consola.

```
// 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- Implemente configuraciones de depósitos, objetos y archivos.

```
// 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 escenarios.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} escenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new escenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
      ${state.retentionBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} escenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>
  new escenarios.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- Ver y eliminar archivos en los cubos.

```
// 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- Destruye todos los recursos creados.

```

// 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 obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for JavaScript .
 - [GetObjectLegalHold](#)
 - [GetObjectLockConfiguration](#)
 - [GetObjectRetención](#)
 - [PutObjectLegalHold](#)
 - [PutObjectLockConfiguration](#)
 - [PutObjectRetención](#)

Cargar o descargar archivos grandes

En el siguiente ejemplo de código se muestra cómo cargar o descargar archivos grandes en y desde Amazon S3.

Para obtener información, consulte [Carga de un objeto con carga multiparte](#).

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cargue un archivo 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);
    }
  }
};
```

Descargue un archivo 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",
  });
};
```

Ejemplos sin servidor

Invocación de una función de Lambda desde un desencadenador de Amazon S3

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al cargar un objeto en un bucket de S3. La función recupera el nombre del bucket de S3 y la clave del objeto del parámetro de evento y llama a la API de Amazon S3 para recuperar y registrar el tipo de contenido del objeto.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de S3 con Lambda mediante JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";
```

```
const client = new S3Client();

exports.handler = async (event, context) => {

  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
  '));

  try {
    const { ContentType } = await client.send(new HeadObjectCommand({
      Bucket: bucket,
      Key: key,
    }));

    console.log('CONTENT TYPE:', ContentType);
    return ContentType;

  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make
    sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

Consumir un evento de S3 con Lambda mediante TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
  '));
```

```
const params = {
  Bucket: bucket,
  Key: key,
};
try {
  const { ContentType } = await s3.send(new HeadObjectCommand(params));
  console.log('CONTENT TYPE:', ContentType);
  return ContentType;
} catch (err) {
  console.log(err);
  const message = `Error getting object ${key} from bucket ${bucket}. Make sure
they exist and your bucket is in the same region as this function.`;
  console.log(message);
  throw new Error(message);
}
};
```

Ejemplos de S3 Glacier que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la AWS SDK for JavaScript (v3) con S3 Glacier.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

CreateVault

En el siguiente ejemplo de código, se muestra cómo usar CreateVault.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 };
```

Cree el almacén.

```
// 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).

- Para obtener más información sobre la API, consulta [CreateVault](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateVault](#) la Referencia AWS SDK for JavaScript de la API.

UploadArchive

En el siguiente ejemplo de código, se muestra cómo usar UploadArchive.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 };
```

Cargue el archivo.

```
// 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [UploadArchive](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [UploadArchive](#) la Referencia AWS SDK for JavaScript de la API.

SageMaker ejemplos de uso del SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con SageMaker

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola SageMaker

Los siguientes ejemplos de código muestran cómo empezar a usarlo SageMaker.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la sección sobre [ListNotebookinstancias](#) en la referencia de la AWS SDK for JavaScript API.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

CreatePipeline

En el siguiente ejemplo de código, se muestra cómo usar CreatePipeline.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Una función que crea una SageMaker canalización mediante una definición de JSON proporcionada 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 obtener más información sobre la API, consulta [CreatePipeline](#) la Referencia AWS SDK for JavaScript de la API.

DeletePipeline

En el siguiente ejemplo de código, se muestra cómo usar DeletePipeline.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

La sintaxis para eliminar una SageMaker canalización. Este código forma parte de una función más amplia. Consulta «Crear una canalización» o el GitHub repositorio para obtener más contexto.

```
await sagemakerClient.send(  
  new DeletePipelineCommand({ PipelineName: name } ),  
);
```

- Para obtener más información sobre la API, consulta la Referencia de [DeletePipeline](#) la AWS SDK for JavaScript API.

DescribePipelineExecution

En el siguiente ejemplo de código, se muestra cómo usar DescribePipelineExecution.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Espere a que la ejecución de una SageMaker canalización tenga éxito, fracase o se detenga.

```
/**  
 * 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 obtener más información sobre la API, consulta [DescribePipeline](#) en la sección [Ejecución](#) en la referencia de la AWS SDK for JavaScript API.

StartPipelineExecution

En el siguiente ejemplo de código, se muestra cómo usar `StartPipelineExecution`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Inicie una ejecución en SageMaker canalización.

```
/**
 * 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 obtener más información sobre la API, consulta [StartPipeline](#) en la sección [Ejecución](#) en la referencia de la AWS SDK for JavaScript API.

Escenarios

Introducción a las tareas y las canalizaciones geoespaciales

En el siguiente ejemplo de código, se muestra cómo:

- Configurar los recursos de una canalización
- Configurar una canalización que ejecuta un trabajo geoespacial
- Iniciar la ejecución de una canalización.
- Supervisar el estado de la ejecución.
- Ver el resultado de la canalización.
- Limpiar recursos.

Para obtener más información, consulte [Crear y ejecutar SageMaker canalizaciones mediante AWS SDK en Community.aws](#).

SDK para (v3) JavaScript

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

El siguiente extracto del archivo contiene funciones que utilizan el SageMaker cliente para gestionar una canalización.

```
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 createEventSourceMapping = () =>  
      lambdaClient.send(  
        new CreateEventSourceMappingCommand({  
          EventSourceArn: queueArn,  
          FunctionName: lambdaName,  
        })),  
      );  
  
    try {  
      const { UUID } = await createEventSourceMapping();  
      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();
}
```

Esta función es un extracto de un archivo que utiliza las funciones de biblioteca anteriores para configurar una SageMaker canalización, ejecutarla y eliminar todos los recursos creados.


```
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 obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for JavaScript .
 - [CreatePipeline](#)
 - [DeletePipeline](#)
 - [DescribePipelineEjecución](#)
 - [StartPipelineEjecución](#)
 - [UpdatePipeline](#)

Ejemplos de Secrets Manager con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante la versión AWS SDK for JavaScript 3 con Secrets Manager.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

GetSecretValue

En el siguiente ejemplo de código, se muestra cómo usar `GetSecretValue`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [GetSecretValue](#) in AWS SDK for JavaScript API Reference.

Ejemplos de Amazon SES que utilizan el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon SES.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas


- [Acciones](#)

Acciones

CreateReceiptFilter

En el siguiente ejemplo de código, se muestra cómo usar `CreateReceiptFilter`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la sección [CreateReceiptFiltrar](#) en la referencia de la AWS SDK for JavaScript API.

CreateReceiptRule

En el siguiente ejemplo de código, se muestra cómo usar CreateReceiptRule.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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,
  emailAddress,
```

```
    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 obtener más información sobre la API, consulta [CreateReceiptla regla](#) en la referencia AWS SDK for JavaScript de la API.

CreateReceiptRuleSet

En el siguiente ejemplo de código, se muestra cómo usar `CreateReceiptRuleSet`.

SDK para JavaScript (v3)

Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateReceiptRuleSet](#) la Referencia AWS SDK for JavaScript de la API.

CreateTemplate

En el siguiente ejemplo de código, se muestra cómo usar `CreateTemplate`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateTemplate](#) la Referencia AWS SDK for JavaScript de la API.

DeleteIdentity

En el siguiente ejemplo de código, se muestra cómo usar DeleteIdentity.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteIdentity](#) la Referencia AWS SDK for JavaScript de la API.

DeleteReceiptFilter

En el siguiente ejemplo de código, se muestra cómo usar DeleteReceiptFilter.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta la sección [DeleteReceiptFiltrar](#) en la referencia de la AWS SDK for JavaScript API.

DeleteReceiptRule

En el siguiente ejemplo de código, se muestra cómo usar DeleteReceiptRule.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteReceiptla regla](#) en la referencia AWS SDK for JavaScript de la API.

DeleteReceiptRuleSet

En el siguiente ejemplo de código, se muestra cómo usar DeleteReceiptRuleSet.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DeleteReceiptRuleSetla Referencia AWS SDK for JavaScript de la API](#).

DeleteTemplate

En el siguiente ejemplo de código, se muestra cómo usar DeleteTemplate.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utills/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 obtener más información sobre la API, consulta [DeleteTemplate](#) la Referencia AWS SDK for JavaScript de la API.

GetTemplate

En el siguiente ejemplo de código, se muestra cómo usar GetTemplate.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [GetTemplate](#) la Referencia AWS SDK for JavaScript de la API.

ListIdentities

En el siguiente ejemplo de código, se muestra cómo usar `ListIdentities`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [ListIdentities](#) la Referencia AWS SDK for JavaScript de la API.

ListReceiptFilters

En el siguiente ejemplo de código, se muestra cómo usar ListReceiptFilters.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).


```
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 obtener más información sobre la API, consulta la [ListReceiptsección Filtros](#) en la referencia de la AWS SDK for JavaScript API.

ListTemplates

En el siguiente ejemplo de código, se muestra cómo usar ListTemplates.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [ListTemplates](#) la Referencia AWS SDK for JavaScript de la API.

SendBulkTemplatedEmail

En el siguiente ejemplo de código, se muestra cómo usar `SendBulkTemplatedEmail`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [SendBulkTemplatedEmail](#) la Referencia AWS SDK for JavaScript de la API.

SendEmail

En el siguiente ejemplo de código, se muestra cómo usar `SendEmail`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [SendEmail](#) en la Referencia AWS SDK for JavaScript de la API.

SendRawEmail

En el siguiente ejemplo de código, se muestra cómo usar `SendRawEmail`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Utilice [nodemailer](#) para enviar un correo electrónico con un archivo adjunto.

```
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 obtener más información sobre la API, consulta el [SendRawcorreo electrónico](#) en la referencia de la AWS SDK for JavaScript API.

SendTemplatedEmail

En el siguiente ejemplo de código, se muestra cómo usar `SendTemplatedEmail`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta el [SendTemplatedcorreo electrónico](#) en la referencia de la AWS SDK for JavaScript API.

UpdateTemplate

En el siguiente ejemplo de código, se muestra cómo usar UpdateTemplate.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [UpdateTemplate](#) la Referencia AWS SDK for JavaScript de la API.

VerifyDomainIdentity

En el siguiente ejemplo de código, se muestra cómo usar `VerifyDomainIdentity`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [VerifyDomainIdentity](#) la referencia sobre la identidad en la AWS SDK for JavaScript API.

VerifyEmailIdentity

En el siguiente ejemplo de código, se muestra cómo usar `VerifyEmailIdentity`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [VerifyEmail](#) [referencia sobre la identidad](#) en la AWS SDK for JavaScript API.

Ejemplos de Amazon SNS con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon SNS.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon SNS

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon SNS.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Inicialice un cliente SNS y muestre los temas de la cuenta.

```
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 obtener más información sobre la API, consulta [ListTopics](#) la Referencia AWS SDK for JavaScript de la API.

Temas

- [Acciones](#)
- [Escenarios](#)
- [Ejemplos sin servidor](#)

Acciones

CheckIfPhoneNumberIsOptedOut

En el siguiente ejemplo de código, se muestra cómo usar CheckIfPhoneNumberIsOptedOut.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CheckIfPhoneNumberIsOptedla referencia de Out](#) in AWS SDK for JavaScript API.

ConfirmSubscription

En el siguiente ejemplo de código, se muestra cómo usar ConfirmSubscription.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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:xxxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
  xxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ConfirmSubscription](#) la Referencia AWS SDK for JavaScript de la API.

CreateTopic

En el siguiente ejemplo de código, se muestra cómo usar CreateTopic.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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:xxxxxxxxxxxxx:TOPIC_NAME'
  // }
}
```

```
    return response;
  };
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateTopic](#) la Referencia AWS SDK for JavaScript de la API.

DeleteTopic

En el siguiente ejemplo de código, se muestra cómo usar DeleteTopic.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteTopic](#) la Referencia AWS SDK for JavaScript de la API.

GetSMSAttributes

En el siguiente ejemplo de código, se muestra cómo usar `GetSMSAttributes`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para ver la información de la API, consulte [GetSMSAttributes](#) en la Referencia de la API de AWS SDK for JavaScript .

GetTopicAttributes

En el siguiente ejemplo de código, se muestra cómo usar GetTopicAttributes.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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: 'xxxxxxxxxxxxx',
//   SubscriptionsPending: '1',
//   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
//   TracingConfig: 'PassThrough',
//   EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetries":0,"headerContentType":"text/plain; charset=UTF-8"}}}',
//   SubscriptionsConfirmed: '0',
//   DisplayName: '',
//   SubscriptionsDeleted: '1'
// }
// }
return response;
};

```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetTopicAttributes](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Importar el SDK y los módulos de cliente, y llamar a la 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 getTopicAttributesPromise = 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetTopic atributos](#) en la referencia AWS SDK for JavaScript de la API.

ListSubscriptions

En el siguiente ejemplo de código, se muestra cómo usar ListSubscriptions.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListSubscriptions](#) la Referencia AWS SDK for JavaScript de la API.

ListTopics

En el siguiente ejemplo de código, se muestra cómo usar ListTopics.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).

- Para obtener más información sobre la API, consulta [ListTopics](#) la Referencia AWS SDK for JavaScript de la API.

Publish

En el siguiente ejemplo de código, se muestra cómo usar Publish.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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-xxxxxxxxxxxxxx'
  // }
  return response;
};
```

Publique un mensaje en un tema con opciones de grupo, duplicación y 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener información sobre la API, consulte [Publicar](#) en la Referencia de la API de AWS SDK for JavaScript .

SetSMSAttributes

En el siguiente ejemplo de código, se muestra cómo usar SetSMSAttributes.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener información sobre la API, consulte [SetSMSAttributes](#) en la Referencia de la API de AWS SDK for JavaScript .

SetTopicAttributes

En el siguiente ejemplo de código, se muestra cómo usar SetTopicAttributes.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [SetTopicAttributes](#) en la referencia AWS SDK for JavaScript de la API.

Subscribe

En el siguiente ejemplo de código, se muestra cómo usar `Subscribe`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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'
  // }
};

```

Suscribe una aplicación móvil a un tema.

```

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;
};

```

Suscriba una función Lambda a un tema.

```

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;  
};
```

Suscriba una cola de SQS a un tema.

```
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-xxxxxxxxxxxxx'  
  // }  
  return response;  
};
```

Suscríbase a un tema con un filtro.

```
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-xxxxxxxxxxxxx'
  // }
  return response;
};
```

- Para obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener información sobre la API, consulte [Suscríbese](#) en la Referencia de la API de AWS SDK for JavaScript .

Unsubscribe

En el siguiente ejemplo de código, se muestra cómo usar Unsubscribe.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurarlo y ejecutarlo en el [Repositorio de ejemplos de código de AWS](#).

Cree el cliente en un módulo separado y expórtelo.

```
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 el SDK y los módulos de cliente, y llame a la 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener información sobre la API, consulte [Cancelar suscripción](#) en la Referencia de la API de AWS SDK for JavaScript .

Escenarios

Publicación de mensajes en colas

En el siguiente ejemplo de código, se muestra cómo:

- Crear un tema (FIFO o no FIFO).
- Suscribirse a varias colas al tema con la opción de aplicar un filtro.
- Publicar mensajes en el tema.
- Sondear las colas en busca de los mensajes recibidos.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este es el punto de entrada de este flujo de trabajo.

```
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();
};
```

El código anterior proporciona las dependencias necesarias e inicia el flujo de trabajo. La siguiente sección contiene la mayor parte del ejemplo.

```
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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAtributos](#)
 - [Publicación](#)
 - [ReceiveMessage](#)
 - [SetQueueAtributos](#)

- [Subscribe](#)
- [Unsubscribe](#)

Ejemplos sin servidor

Invocar una función de Lambda desde un desencadenador de Amazon SNS

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al recibir mensajes de un tema de SNS. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de SNS con JavaScript Lambda mediante.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumir un evento de SNS con TypeScript Lambda mediante.

```
// 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;
  }
}
```

Ejemplos de Amazon SQS con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon SQS.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon SQS

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon SQS.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Inicializar un cliente de Amazon SQS y enumerar las colas.

```
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 obtener más información sobre la API, consulta [ListQueues](#) la Referencia AWS SDK for JavaScript de la API.

Temas

- [Acciones](#)
- [Escenarios](#)
- [Ejemplos sin servidor](#)

Acciones

ChangeMessageVisibility

En el siguiente ejemplo de código, se muestra cómo usar ChangeMessageVisibility.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Recibir un mensaje de Amazon SQS y cambiar la visibilidad del tiempo de espera.

```
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 obtener más información sobre la API, consulta [ChangeMessageVisibilidad](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Recibir un mensaje de Amazon SQS y cambiar la visibilidad del tiempo de espera.

```
// 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ChangeMessageVisibilidad](#) en la referencia AWS SDK for JavaScript de la API.

CreateQueue

En el siguiente ejemplo de código, se muestra cómo usar CreateQueue.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Crear una cola estándar de 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;
};
```

Crear una cola de Amazon SQS con sondeo largo.

```
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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [CreateQueue](#) en la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Crear una cola estándar de 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);
  }
});
```

Crear una cola de Amazon SQS que espere a que llegue un mensaje.

```
// 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).

- Para obtener más información sobre la API, consulta [CreateQueue](#) la Referencia AWS SDK for JavaScript de la API.

DeleteMessage

En el siguiente ejemplo de código, se muestra cómo usar DeleteMessage.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Recibir y eliminar mensajes de 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 obtener más información sobre la API, consulta [DeleteMessage](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Recibir y eliminar mensajes de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteMessage](#) la Referencia AWS SDK for JavaScript de la API.

DeleteMessageBatch

En el siguiente ejemplo de código, se muestra cómo usar DeleteMessageBatch.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulte [DeleteMessageBatch](#) in AWS SDK for JavaScript API Reference.

DeleteQueue

En el siguiente ejemplo de código, se muestra cómo usar DeleteQueue.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Eliminar una cola de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteQueue](#) en la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Eliminar una cola de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteQueue](#) la Referencia AWS SDK for JavaScript de la API.

GetQueueAttributes

En el siguiente ejemplo de código, se muestra cómo usar GetQueueAttributes.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [GetQueueAttributes](#) en la referencia AWS SDK for JavaScript de la API.

GetQueueUrl

En el siguiente ejemplo de código, se muestra cómo usar `GetQueueUrl`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtener la URL de una cola de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetQueueUrl](#) en la referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Obtener la URL de una cola de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [GetQueueUrl](#) en la referencia AWS SDK for JavaScript de la API.

ListQueues

En el siguiente ejemplo de código, se muestra cómo usar ListQueues.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumerar las colas de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListQueues](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumerar las colas de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListQueues](#) la Referencia AWS SDK for JavaScript de la API.

ReceiveMessage

En el siguiente ejemplo de código, se muestra cómo usar ReceiveMessage.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Recibir un mensaje de una cola de 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,
        })),
      }),
    );
  }
}
```


```
    );  
  }  
};
```

Recibir un mensaje de una cola de Amazon SQS mediante el soporte de sondeos largos.

```
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 obtener más información sobre la API, consulta [ReceiveMessage](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

 Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Recibir un mensaje de una cola de Amazon SQS mediante el soporte de sondeos largos.

```
// 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ReceiveMessage](#) la Referencia AWS SDK for JavaScript de la API.

SendMessage

En el siguiente ejemplo de código, se muestra cómo usar SendMessage.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enviar un mensaje a una cola de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [SendMessage](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enviar un mensaje a una cola de 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [SendMessage](#) la Referencia AWS SDK for JavaScript de la API.

SetQueueAttributes

En el siguiente ejemplo de código, se muestra cómo usar SetQueueAttributes.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 una cola de Amazon SQS para utilizar sondeos largos.

```
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 una cola de mensajes fallidos.

```
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 obtener más información sobre la API, consulta [SetQueueAttributes](#) en la referencia AWS SDK for JavaScript de la API.

Escenarios

Publicación de mensajes en colas

En el siguiente ejemplo de código, se muestra cómo:

- Crear un tema (FIFO o no FIFO).
- Suscribirse a varias colas al tema con la opción de aplicar un filtro.
- Publicar mensajes en el tema.
- Sondar las colas en busca de los mensajes recibidos.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este es el punto de entrada de este flujo de trabajo.

```
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();
};
```

El código anterior proporciona las dependencias necesarias e inicia el flujo de trabajo. La siguiente sección contiene la mayor parte del ejemplo.

```
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 obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .

- [CreateQueue](#)
- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAtributos](#)
- [Publicación](#)
- [ReceiveMessage](#)
- [SetQueueAtributos](#)
- [Subscribe](#)
- [Unsubscribe](#)

Ejemplos sin servidor

Invocar una función de Lambda desde un desencadenador de Amazon SQS

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al recibir mensajes de una cola de SQS. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

SDK para JavaScript (v3)

Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de SQS con JavaScript Lambda mediante.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
}
```

```
    console.info("done");
  };

  async function processMessageAsync(message) {
    try {
      console.log(`Processed message ${message.body}`);
      // TODO: Do interesting work based on the new message
      await Promise.resolve(1); //Placeholder for actual async work
    } catch (err) {
      console.error("An error occurred");
      throw err;
    }
  }
}
```

Consumir un evento de SQS con TypeScript Lambda mediante.

```
// 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;
  }
}
```


Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS.

En el siguiente ejemplo de código se muestra cómo implementar una respuesta por lotes parcial para funciones de Lambda que reciben eventos de una cola de SQS. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Informe de errores de artículos de lotes de SQS con JavaScript Lambda mediante.

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

Informe de errores de artículos de lotes de SQS con TypeScript Lambda mediante.

```
// 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.');
```

Ejemplos de Step Functions con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Step Functions.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

StartExecution

En el siguiente ejemplo de código, se muestra cómo usar StartExecution.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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:aaaaaaa-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 obtener más información sobre la API, consulta [StartExecution](#) la Referencia AWS SDK for JavaScript de la API.

AWS STS ejemplos de uso del SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con AWS STS

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

AssumeRole

En el siguiente ejemplo de código, se muestra cómo usar AssumeRole.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 });
```

Asuma el rol de 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 obtener más información sobre la API, consulta [AssumeRole](#) la Referencia AWS SDK for JavaScript de la API.

SDK para JavaScript (v2)

Note

Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// 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 obtener más información sobre la API, consulta [AssumeRole](#) la Referencia AWS SDK for JavaScript de la API.

AWS Support ejemplos de uso del SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con. AWS Support

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola AWS Support

En los siguientes ejemplos de código se muestra cómo empezar a utilizar AWS Glue.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

invoque 'main()' para ejecutar el ejemplo.

```
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 obtener más información sobre la API, consulta [DescribeServices](#) la Referencia AWS SDK for JavaScript de la API.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

AddAttachmentsToSet

En el siguiente ejemplo de código, se muestra cómo usar AddAttachmentsToSet.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [AddAttachmentsToSet](#) la Referencia AWS SDK for JavaScript de la API.

AddCommunicationToCase

En el siguiente ejemplo de código, se muestra cómo usar `AddCommunicationToCase`.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [AddCommunicationToCase](#) la Referencia AWS SDK for JavaScript de la API.

CreateCase

En el siguiente ejemplo de código, se muestra cómo usar CreateCase.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [CreateCase](#) la Referencia AWS SDK for JavaScript de la API.

DescribeAttachment

En el siguiente ejemplo de código, se muestra cómo usar DescribeAttachment.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DescribeAttachment](#) la Referencia AWS SDK for JavaScript de la API.

DescribeCases

En el siguiente ejemplo de código, se muestra cómo usar DescribeCases.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DescribeCases](#) la Referencia AWS SDK for JavaScript de la API.

DescribeCommunications

En el siguiente ejemplo de código, se muestra cómo usar DescribeCommunications.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [DescribeCommunications](#) la Referencia AWS SDK for JavaScript de la API.

DescribeSeverityLevels

En el siguiente ejemplo de código, se muestra cómo usar DescribeSeverityLevels.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener información detallada sobre la API, consulta la [DescribeSeveritysección Niveles](#) in AWS SDK for JavaScript API Reference.

ResolveCase

En el siguiente ejemplo de código, se muestra cómo usar ResolveCase.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
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 obtener más información sobre la API, consulta [ResolveCase](#) la Referencia AWS SDK for JavaScript de la API.

Escenarios

Introducción a los casos

En el siguiente ejemplo de código, se muestra cómo:

- Obtenga y muestre los servicios disponibles y los niveles de gravedad de los casos.
- Cree un caso de asistencia mediante un servicio, una categoría y un nivel de gravedad seleccionados.
- Obtenga y muestre una lista de casos abiertos para el día actual.
- Añada una serie de archivos adjuntos y una comunicación al nuevo caso.
- Describa el nuevo archivo adjunto y la comunicación del caso.
- Resuelva el caso.
- Obtenga y muestre una lista de casos resueltos para el día actual.

SDK para JavaScript (v3)

Note

Hay más información. [GitHub](#) Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecute un escenario interactivo en el 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: startOfDay.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 },
      () => getTodayResolvedCases(caseId),
    );
    console.log("Resolved cases:");
    console.log(resolvedCases.map((c) => c.caseId).join("\n"));
  }
} catch (err) {
  console.error(err);
}
};
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript .
 - [AddAttachmentsToSet](#)
 - [AddCommunicationToCase](#)
 - [CreateCase](#)
 - [DescribeAttachment](#)
 - [DescribeCases](#)
 - [DescribeCommunications](#)
 - [DescribeServices](#)
 - [DescribeSeverityNiveles](#)
 - [ResolveCase](#)

Ejemplos de Amazon Transcribe con el SDK para JavaScript (v3)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for JavaScript (v3) con Amazon Transcribe.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- [Acciones](#)

Acciones

DeleteMedicalTranscriptionJob

En el siguiente ejemplo de código, se muestra cómo usar DeleteMedicalTranscriptionJob.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 };
```

Eliminar un trabajo de transcripción 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_transcription_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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteMedicalTranscriptionJob](#) Referencia AWS SDK for JavaScript de la API.

DeleteTranscriptionJob

En el siguiente ejemplo de código, se muestra cómo usar DeleteTranscriptionJob.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Eliminar un trabajo de transcripción.

```
// 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, 'transcription_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();
```

Cree el 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [DeleteTranscriptionJob](#) in AWS SDK for JavaScript API Reference.

ListMedicalTranscriptionJobs

En el siguiente ejemplo de código, se muestra cómo usar `ListMedicalTranscriptionJobs`.

SDK para JavaScript (v3)

 Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 };
```

Enumerar trabajos de transcripción 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [ListMedicalTranscriptionJobs](#) la Referencia AWS SDK for JavaScript de la API.

ListTranscriptionJobs

En el siguiente ejemplo de código, se muestra cómo usar ListTranscriptionJobs.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumerar trabajos de transcripción.

```
// 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();
```

Cree el 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta la [ListTranscriptionsección Jobs](#) in AWS SDK for JavaScript API Reference.

StartMedicalTranscriptionJob

En el siguiente ejemplo de código, se muestra cómo usar StartMedicalTranscriptionJob.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree el 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 un trabajo de transcripción 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [StartMedicalTranscriptionJob](#) la Referencia AWS SDK for JavaScript de la API.

StartTranscriptionJob

En el siguiente ejemplo de código, se muestra cómo usar StartTranscriptionJob.

SDK para JavaScript (v3)

Note

Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Iniciar un trabajo de transcripción.

```
// 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();
```

Cree el 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 obtener información, consulte la [Guía para desarrolladores de AWS SDK for JavaScript](#).
- Para obtener más información sobre la API, consulta [StartTranscriptionJob](#) in AWS SDK for JavaScript API Reference.

Ejemplos de servicios cruzados que utilizan el SDK para JavaScript (v3)

Las siguientes aplicaciones de ejemplo utilizan la AWS SDK for JavaScript (v3) para trabajar en múltiples aplicaciones. Servicios de AWS

Los ejemplos de servicios combinados apuntan a un nivel avanzado de experiencia para ayudarle a empezar a crear aplicaciones.

Ejemplos

- [Cree una aplicación Amazon Transcribe](#)
- [Creación de una aplicación de streaming de Amazon Transcribe](#)
- [Creación de una aplicación para enviar datos a una tabla de DynamoDB](#)

- [Crear un chatbot de Amazon Lex para atraer visitantes a su sitio web](#)
- [Creación de una aplicación de administración de activos fotográficos que permita a los usuarios administrar las fotos mediante etiquetas](#)
- [Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB](#)
- [Crear un rastreador de elementos de trabajo de Aurora Serverless](#)
- [Creación de una aplicación de exploración de Amazon Textract](#)
- [Creación de una aplicación que analice los comentarios de los clientes y sintetice el audio](#)
- [Detecte PPE en imágenes con Amazon Rekognition AWS mediante un SDK](#)
- [Detecte objetos en imágenes con Amazon Rekognition AWS mediante un SDK](#)
- [Detecte personas y objetos en un vídeo con Amazon Rekognition AWS mediante un SDK](#)
- [Invocación de una función de Lambda desde un navegador](#)
- [Uso de API Gateway para invocar una función de Lambda](#)
- [Uso de Step Functions para invocar funciones de Lambda](#)
- [Usar eventos programados para invocar una función de Lambda](#)

Cree una aplicación Amazon Transcribe

SDK para JavaScript (v3)

Cree una aplicación que utilice Amazon Transcribe para transcribir y mostrar grabaciones de voz en el navegador. La aplicación utiliza dos buckets de Amazon Simple Storage Service (Amazon S3), uno para alojar el código de la aplicación y otro para almacenar transcripciones. La aplicación utiliza un grupo de usuarios de Amazon Cognito para autenticar a los usuarios. Los usuarios autenticados tienen permisos AWS Identity and Access Management (IAM) para acceder a los servicios necesarios. AWS

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en [GitHub](#)

Este ejemplo también está disponible en la [guía para desarrolladores de AWS SDK for JavaScript v3](#).

Servicios utilizados en este ejemplo

- Amazon Cognito Identity

- Amazon S3
- Amazon Transcribe

Creación de una aplicación de streaming de Amazon Transcribe

SDK para JavaScript (v3)

Muestra cómo utilizar Amazon Transcribe para crear una aplicación que grabe, transcriba y traduzca audio en directo en tiempo real para luego enviar los resultados por correo electrónico mediante Amazon Simple Email Service (Amazon SES).

Para ver el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Creación de una aplicación para enviar datos a una tabla de DynamoDB

SDK para JavaScript (v3)

Este ejemplo indica cómo crear una aplicación que permita a los usuarios enviar datos a una tabla de Amazon DynamoDB y un mensaje de texto al administrador mediante Amazon Simple Notification Service (Amazon SNS).

Para ver el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en [GitHub](#).

Este ejemplo también está disponible en la [guía para desarrolladores de AWS SDK for JavaScript v3](#).

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SNS

Crear un chatbot de Amazon Lex para atraer visitantes a su sitio web

SDK para JavaScript (v3)

Indica cómo utilizar la API de Amazon Lex para crear un Chatbot dentro de una aplicación web con el fin de atraer visitantes a su sitio web.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo sobre cómo [crear un chatbot Amazon Lex](#) en la guía para AWS SDK for JavaScript desarrolladores.

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Creación de una aplicación de administración de activos fotográficos que permita a los usuarios administrar las fotos mediante etiquetas

SDK para JavaScript (v3)

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en [GitHub](#).

Para profundizar en el origen de este ejemplo, consulte la publicación en [Comunidad de AWS](#).

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB

SDK para JavaScript (v3)

Muestra cómo utilizar la API de Amazon DynamoDB para crear una aplicación web dinámica que haga un seguimiento de los datos de trabajo de DynamoDB.

Para ver el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SES

Crear un rastreador de elementos de trabajo de Aurora Serverless

SDK para JavaScript (v3)

Muestra cómo utilizar la versión AWS SDK for JavaScript 3 para crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos de Amazon Aurora y envíe informes por correo electrónico mediante Amazon Simple Email Service (Amazon SES). Este ejemplo usa un frontend creado con React.js para interactuar con un backend de Node.js de Express.

- Integre una aplicación web React.js con Servicios de AWS.
- Cree una lista, agregue y actualice elementos en una tabla de Aurora.
- Envíe un informe por correo electrónico de elementos de trabajo filtrados con Amazon SES.
- Implemente y gestione recursos de ejemplo con el AWS CloudFormation script incluido.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- Aurora
- Amazon RDS
- Servicio de datos de Amazon RDS

- Amazon SES

Creación de una aplicación de exploración de Amazon Textract

SDK para JavaScript (v3)

Muestra cómo utilizarla AWS SDK for JavaScript para crear una aplicación de React que utilice Amazon Textract para extraer datos de la imagen de un documento y mostrarlos en una página web interactiva. Este ejemplo se ejecuta en un navegador web y requiere una identidad autenticada de Amazon Cognito para las credenciales. Para el almacenamiento utiliza Amazon Simple Storage Service (Amazon S3) y para las notificaciones consulta una cola de Amazon Simple Queue Service (Amazon SQS) que está suscrita a un tema de Amazon Simple Notification Service (Amazon SNS).

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Creación de una aplicación que analice los comentarios de los clientes y sintetice el audio

SDK para JavaScript (v3)

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.

- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en [GitHub](#). Los siguientes extractos muestran cómo AWS SDK for JavaScript se usa dentro de las funciones 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 };
};
```

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Detecte PPE en imágenes con Amazon Rekognition AWS mediante un SDK

SDK para (v3 JavaScript)

Muestra cómo utilizar Amazon Rekognition AWS SDK for JavaScript con el para crear una aplicación que detecte el equipo de protección personal (EPP) en imágenes ubicadas en un depósito de Amazon Simple Storage Service (Amazon S3). La aplicación guarda los resultados en una tabla de Amazon DynamoDB y envía al administrador una notificación por correo electrónico con los resultados mediante Amazon Simple Email Service (Amazon SES).

Aprenda cómo:

- Crear un usuario no autenticado con Amazon Cognito.
- Analizar imágenes en busca de EPI con Amazon Rekognition.
- Verificar una dirección de correo electrónico de Amazon SES.
- Actualizar una tabla de DynamoDB con resultados.
- Enviar una notificación por correo electrónico con Amazon SES.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en. [GitHub](#)

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Detecte objetos en imágenes con Amazon Rekognition AWS mediante un SDK

SDK para (v3 JavaScript)

Muestra cómo utilizar Amazon Rekognition AWS SDK for JavaScript con el para crear una aplicación que utilice Amazon Rekognition para identificar objetos por categoría en imágenes ubicadas en un bucket de Amazon Simple Storage Service (Amazon S3). La aplicación envía al administrador una notificación por correo electrónico con los resultados mediante Amazon Simple Email Service (Amazon SES).

Aprenda cómo:

- Crear un usuario no autenticado con Amazon Cognito.
- Analizar imágenes en busca de objetos con Amazon Rekognition.
- Verificar una dirección de correo electrónico de Amazon SES.
- Enviar una notificación por correo electrónico con Amazon SES.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en. [GitHub](#)

Servicios utilizados en este ejemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Detecte personas y objetos en un vídeo con Amazon Rekognition AWS mediante un SDK

SDK para (v3 JavaScript)

Muestra cómo usar Amazon Rekognition AWS SDK for JavaScript con el para crear una aplicación que detecte rostros y objetos en vídeos ubicados en un bucket de Amazon Simple Storage Service (Amazon S3). La aplicación envía al administrador una notificación por correo electrónico con los resultados mediante Amazon Simple Email Service (Amazon SES).

Aprenda cómo:

- Crear un usuario no autenticado con Amazon Cognito.
- Analizar imágenes en busca de EPI con Amazon Rekognition.
- Verificar una dirección de correo electrónico de Amazon SES.
- Enviar una notificación por correo electrónico con Amazon SES.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en. [GitHub](#)

Servicios utilizados en este ejemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Invocación de una función de Lambda desde un navegador

SDK para JavaScript (v2)

Puede crear una aplicación basada en un navegador que utilice una AWS Lambda función para actualizar una tabla de Amazon DynamoDB con las selecciones de los usuarios.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en [GitHub](#)

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda

SDK para JavaScript (v3)

Puede crear una aplicación basada en un navegador que utilice una AWS Lambda función para actualizar una tabla de Amazon DynamoDB con las selecciones de los usuarios. Esta aplicación utiliza la versión 3. AWS SDK for JavaScript

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda

Uso de API Gateway para invocar una función de Lambda

SDK para JavaScript (v3)

Muestra cómo crear una AWS Lambda función mediante la API de tiempo de JavaScript ejecución de Lambda. En este ejemplo, se invocan diferentes AWS servicios para realizar un caso de uso específico. En este ejemplo se indica cómo crear una función de Lambda invocada por Amazon API Gateway que escanea una tabla de Amazon DynamoDB en busca de aniversarios laborales y utiliza Amazon Simple Notification Service (Amazon SNS) para enviar un mensaje de texto a sus empleados que les felicite en la fecha de su primer aniversario.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en [GitHub](#).

Este ejemplo también está disponible en la [guía para desarrolladores de AWS SDK for JavaScript v3](#).

Servicios utilizados en este ejemplo

- API Gateway

- DynamoDB
- Lambda
- Amazon SNS

Uso de Step Functions para invocar funciones de Lambda

SDK para JavaScript (v3)

Muestra cómo crear un flujo de trabajo AWS sin servidor mediante AWS Step Functions y el AWS SDK for JavaScript. Cada paso del flujo de trabajo se implementa mediante una AWS Lambda función.

Lambda es un servicio de computación que permite ejecutar código sin aprovisionar ni administrar servidores. Step Functions es un servicio de orquestación sin servidor que le permite combinar funciones de Lambda y otros servicios de AWS para crear aplicaciones esenciales desde el punto de vista empresarial.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en [GitHub](#).

Este ejemplo también está disponible en la [guía para desarrolladores de AWS SDK for JavaScript v3](#).

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Usar eventos programados para invocar una función de Lambda

SDK para JavaScript (v3)

Muestra cómo crear un evento EventBridge programado de Amazon que invoque una AWS Lambda función. Configure EventBridge para usar una expresión cron para programar cuándo se invoca la función Lambda. En este ejemplo, se crea una función Lambda mediante la API de tiempo de ejecución de JavaScript Lambda. En este ejemplo, se invocan diferentes AWS

servicios para realizar un caso de uso específico. Este ejemplo indica cómo crear una aplicación que envíe un mensaje de texto a sus empleados para felicitarles por su primer aniversario.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en [GitHub](#).

Este ejemplo también está disponible en la [guía para desarrolladores de AWS SDK for JavaScript v3](#).

Servicios utilizados en este ejemplo

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Seguridad para este AWS producto o servicio

La seguridad en la nube de Amazon Web Services (AWS) es la máxima prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos que se han diseñado para satisfacer los requisitos de seguridad de las organizaciones más exigentes. La seguridad es una responsabilidad compartida entre usted AWS y usted. En el [modelo de responsabilidad compartida](#), se habla de “seguridad de la nube” y “seguridad en la nube”:

Seguridad de la nube: AWS se encarga de proteger la infraestructura en la que se ejecutan todos los servicios que se ofrecen en la AWS nube y de proporcionarle servicios que pueda utilizar de forma segura. Nuestra responsabilidad en materia de seguridad es nuestra máxima prioridad AWS, y auditores externos comprueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los [programas de AWS conformidad](#).

Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice y otros factores, como la confidencialidad de sus datos, los requisitos de su organización y las leyes y reglamentos aplicables.

Este AWS producto o servicio sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la [página de documentación sobre la seguridad del AWS servicio](#) y [AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad](#).

Temas

- [Protección de datos en este AWS producto o servicio](#)
- [Identity and Access Management](#)
- [Validación de la conformidad de este AWS producto o servicio](#)
- [Resiliencia de este AWS producto o servicio](#)
- [Seguridad de la infraestructura para este AWS producto o servicio](#)
- [Aplicar una versión mínima de TLS](#)

Protección de datos en este AWS producto o servicio

El [modelo de](#) se aplica a protección de datos de este AWS producto o servicio. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global en la que se ejecutan

todos los Nube de AWS. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Usted también es responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las [Preguntas frecuentes sobre la privacidad de datos](#). Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el [Modelo de responsabilidad compartida de AWS y GDPR](#) en el Blog de seguridad de AWS .

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure los usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utilice la autenticación multifactor (MFA) en cada cuenta.
- Utilice SSL/TLS para comunicarse con los recursos. AWS Se recomienda el uso de TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad de los usuarios con. AWS CloudTrail
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita módulos criptográficos validados por FIPS 140-2 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte [Estándar de procesamiento de la información federal \(FIPS\) 140-2](#).

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como, por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabaja con este AWS producto o servicio u otro Servicios de AWS mediante la consola, la API o los SDK. AWS CLI AWS Cualquier dato que ingrese en etiquetas o campos de formato libre utilizados para nombres se puede emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Identity and Access Management

AWS Identity and Access Management (IAM) es una herramienta Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a los AWS recursos. Los administradores de IAM controlan quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar los recursos. AWS La IAM es una Servicio de AWS opción que puede utilizar sin coste adicional.

Temas

- [Público](#)
- [Autenticación con identidades](#)
- [Administración de acceso mediante políticas](#)
- [¿Cómo Servicios de AWS trabajar con IAM](#)
- [Solución de problemas de AWS identidad y acceso](#)

Público

La forma de usar AWS Identity and Access Management (IAM) varía según el trabajo en el que se realice. AWS

Usuario del servicio: si Servicios de AWS solía hacer su trabajo, el administrador le proporcionará las credenciales y los permisos que necesita. A medida que vaya utilizando más AWS funciones para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una función de AWS, consulte [Solución de problemas de AWS identidad y acceso](#) o consulte la guía del usuario de la Servicio de AWS que está utilizando.

Administrador de servicios: si está a cargo de AWS los recursos de su empresa, probablemente tenga acceso total a ellos AWS. Su trabajo consiste en determinar a qué AWS funciones y recursos deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo su empresa puede utilizar la IAM AWS, consulte la guía del usuario del Servicio de AWS que está utilizando.

Administrador de IAM: si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a AWS. Para ver ejemplos de políticas AWS

basadas en la identidad que puede utilizar en IAM, consulte la guía del usuario de la Servicio de AWS que está utilizando.

Autenticación con identidades

La autenticación es la forma de iniciar sesión AWS con sus credenciales de identidad. Debe estar autenticado (con quien haya iniciado sesión AWS) como usuario de IAM o asumiendo una función de IAM. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (Centro de identidades de IAM), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su administrador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accedes AWS mediante la federación, estás asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte [Cómo iniciar sesión Cuenta de AWS en su](#) Guía del AWS Sign-In usuario.

Si accede AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener más información sobre cómo usar el método recomendado para firmar las solicitudes usted mismo, consulte [Firmar las solicitudes de la AWS API](#) en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte [Autenticación multifactor](#) en la Guía del usuario de AWS IAM Identity Center y [Uso de la autenticación multifactor \(MFA\) en AWS](#) en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos Servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren

que inicie sesión como usuario raíz, consulte [Tareas que requieren credenciales de usuario raíz](#) en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos, incluidos los que requieren acceso de administrador, que utilicen la federación con un proveedor de identidades para acceder Servicios de AWS mediante credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidades web AWS Directory Service, el directorio del Centro de Identidad o cualquier usuario al que acceda Servicios de AWS mediante las credenciales proporcionadas a través de una fuente de identidad. Cuando las identidades federadas acceden Cuentas de AWS, asumen funciones y las funciones proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utilice AWS IAM Identity Center. Puede crear usuarios y grupos en el Centro de identidades de IAM o puede conectarse y sincronizarse con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todas sus Cuentas de AWS aplicaciones. Para obtener más información, consulte [¿Qué es el Centro de identidades de IAM?](#) en la Guía del usuario de AWS IAM Identity Center .

Usuarios y grupos de IAM

Un [usuario de IAM](#) es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte [Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración](#) en la Guía del usuario de IAM.

Un [grupo de IAM](#) es una identidad que especifica un conjunto de usuarios de IAM. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales.

Para más información, consulte [Cuándo crear un usuario de IAM \(en lugar de un rol\)](#) en la Guía del usuario de IAM.

Roles de IAM

Un [rol de IAM](#) es una identidad dentro de usted Cuenta de AWS que tiene permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una determinada persona. Puede asumir temporalmente una función de IAM en el AWS Management Console [cambiando](#) de función. Puede asumir un rol llamando a una operación de AWS API AWS CLI o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulte [Uso de roles de IAM](#) en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- **Acceso de usuario federado:** para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles para federación, consulte [Creación de un rol para un proveedor de identidades de terceros](#) en la Guía del usuario de IAM. Si utiliza IAM Identity Center, debe configurar un conjunto de permisos. IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulte [Conjuntos de permisos](#) en la Guía del usuario de AWS IAM Identity Center .
- **Permisos de usuario de IAM temporales:** un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- **Acceso entre cuentas:** puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunas Servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte [Cómo los roles de IAM difieren de las políticas basadas en recursos](#) en la Guía del usuario de IAM.
- **Acceso entre servicios:** algunos Servicios de AWS utilizan funciones en otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.

- **Sesiones de acceso directo (FAS):** cuando utilizas un usuario o un rol de IAM para realizar acciones en ellas AWS, se te considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. El FAS utiliza los permisos del principal que llama Servicio de AWS y los solicita Servicio de AWS para realizar solicitudes a los servicios descendentes. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte [Reenviar sesiones de acceso](#).
- **Rol de servicio:** un rol de servicio es un [rol de IAM](#) que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte [Creación de un rol para delegar permisos a un Servicio de AWS](#) en la Guía del usuario de IAM.
- **Función vinculada al servicio:** una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.
- **Aplicaciones que se ejecutan en Amazon EC2:** puede usar un rol de IAM para administrar las credenciales temporales de las aplicaciones que se ejecutan en una instancia EC2 y realizan AWS CLI solicitudes a la API. AWS Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia de EC2. Para asignar una AWS función a una instancia EC2 y ponerla a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia de EC2 obtener credenciales temporales. Para más información, consulte [Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias Amazon EC2](#) en la Guía del usuario de IAM.

Para obtener información sobre el uso de los roles de IAM, consulte [Cuándo crear un rol de IAM \(en lugar de un usuario\)](#) en la Guía del usuario de IAM.

Administración de acceso mediante políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una

solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan en AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte [Información general de políticas JSON](#) en la Guía del usuario de IAM.

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre el rol de la API AWS Management Console, la AWS CLI, o la AWS API.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte [Creación de políticas de IAM](#) en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y roles de su Cuenta de AWS empresa. Las políticas administradas incluyen políticas administradas y políticas administradas por el cliente. Para más información sobre cómo elegir una política administrada o una política insertada, consulte [Elegir entre políticas administradas y políticas insertadas](#) en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico.

Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe [especificar una entidad principal](#) en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar políticas AWS gestionadas de IAM en una política basada en recursos.

Listas de control de acceso (ACL)

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3 y Amazon VPC son ejemplos de servicios que admiten las ACL. AWS WAF Para obtener más información sobre las ACL, consulte [Información general de Lista de control de acceso \(ACL\)](#) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales y menos comunes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- **Límites de permisos:** un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte [Límites de permisos para las entidades de IAM](#) en la Guía del usuario de IAM.
- **Políticas de control de servicios (SCP):** las SCP son políticas de JSON que especifican los permisos máximos para una organización o unidad organizativa (OU). AWS Organizations AWS Organizations es un servicio para agrupar y gestionar de forma centralizada varios de los Cuentas de AWS que son propiedad de su empresa. Si habilita todas las características en una organización, entonces podrá aplicar políticas de control de servicio (SCP) a una o a todas sus cuentas. El SCP limita los permisos de las entidades en las cuentas de los miembros, incluidas las de cada una. Usuario raíz de la cuenta de AWS Para obtener más información acerca de

Organizations y las SCP, consulte [Funcionamiento de las SCP](#) en la Guía del usuario de AWS Organizations .

- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte [Políticas de sesión](#) en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo AWS determinar si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la [lógica de evaluación de políticas](#) en la Guía del usuario de IAM.

¿Cómo Servicios de AWS trabajar con IAM

Para obtener una visión general de cómo Servicios de AWS trabajar con la mayoría de las funciones de IAM, consulte [AWS los servicios que funcionan con IAM en la Guía del usuario de IAM](#).

Para obtener información sobre cómo utilizar una función específica Servicio de AWS con IAM, consulte la sección de seguridad de la guía del usuario del servicio correspondiente.

Solución de problemas de AWS identidad y acceso

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas habituales que pueden surgir al trabajar con un AWS IAM.

Temas

- [No estoy autorizado a realizar ninguna acción en AWS](#)
- [No estoy autorizado a realizar tareas como: PassRole](#)
- [Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS recursos](#)

No estoy autorizado a realizar ninguna acción en AWS

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM `mateojackson` intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio `my-example-widget`, pero no tiene los permisos ficticios `awes:GetWidget`.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
awes:GetWidget on resource: my-example-widget
```

En este caso, la política del usuario `mateojackson` debe actualizarse para permitir el acceso al recurso `my-example-widget` mediante la acción `awes:GetWidget`.

Si necesita ayuda, póngase en contacto con su AWS administrador. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

No estoy autorizado a realizar tareas como: PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a AWS.

Algunos Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada a un servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en AWS. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su administrador. AWS El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS recursos

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que

asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para más información, consulte lo siguiente:

- Para saber si AWS es compatible con estas funciones, consulte [¿Cómo Servicios de AWS trabajar con IAM.](#)
- Para obtener información sobre cómo proporcionar acceso a los recursos de su Cuentas de AWS propiedad, consulte [Proporcionar acceso a un usuario de IAM en otro usuario de su propiedad Cuenta de AWS en](#) la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta [Cómo proporcionar acceso a recursos que Cuentas de AWS son propiedad de terceros](#) en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulte [Proporcionar acceso a usuarios autenticados externamente \(identidad federada\)](#) en la Guía del usuario de IAM.
- Para obtener información sobre la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte [Cómo los roles de IAM difieren de las políticas basadas en recursos](#) en la Guía del usuario de IAM.


Validación de la conformidad de este AWS producto o servicio

Para saber si un programa de cumplimiento Servicio de AWS está dentro del ámbito de aplicación de programas de cumplimiento específicos, consulte [Servicios de AWS Alcance por programa](#) de de cumplimiento y elija el programa de cumplimiento que le interese. Para obtener información general, consulte Programas de [AWS cumplimiento > Programas AWS](#) .

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte [Descarga de informes en AWS Artifact](#) .

Su responsabilidad de cumplimiento al Servicios de AWS utilizarlos viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. AWS proporciona los siguientes recursos para ayudar con el cumplimiento:

- [Guías de inicio rápido sobre seguridad y cumplimiento](#): estas guías de implementación analizan las consideraciones arquitectónicas y proporcionan los pasos para implementar entornos básicos centrados en AWS la seguridad y el cumplimiento.
- Diseño de [arquitectura para garantizar la seguridad y el cumplimiento de la HIPAA en Amazon Web Services](#): en este documento técnico se describe cómo pueden utilizar AWS las empresas para crear aplicaciones aptas para la HIPAA.

 Note

No Servicios de AWS todas cumplen los requisitos de la HIPAA. Para más información, consulte la [Referencia de servicios compatibles con HIPAA](#).

- [AWS Recursos de](#) de cumplimiento: esta colección de libros de trabajo y guías puede aplicarse a su industria y ubicación.
- [AWS Guías de cumplimiento para clientes](#): comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. Las guías resumen las mejores prácticas para garantizar la seguridad Servicios de AWS y orientan los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST), el Consejo de Normas de Seguridad del Sector de Tarjetas de Pago (PCI) y la Organización Internacional de Normalización (ISO)).
- [Evaluación de los recursos con reglas](#) en la guía para AWS Config desarrolladores: el AWS Config servicio evalúa en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normas.
- [AWS Security Hub](#)— Esto Servicio de AWS proporciona una visión completa del estado de su seguridad interior AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su cumplimiento con los estándares y las prácticas recomendadas del sector de la seguridad. Para obtener una lista de los servicios y controles compatibles, consulte la [Referencia de controles de Security Hub](#).
- [Amazon GuardDuty](#): Servicio de AWS detecta posibles amenazas para sus cargas de trabajo Cuentas de AWS, contenedores y datos mediante la supervisión de su entorno para detectar actividades sospechosas y maliciosas. GuardDuty puede ayudarlo a cumplir con varios requisitos de conformidad, como el PCI DSS, al cumplir con los requisitos de detección de intrusiones exigidos por ciertos marcos de cumplimiento.
- [AWS Audit Manager](#)— Esto le Servicio de AWS ayuda a auditar continuamente su AWS uso para simplificar la gestión del riesgo y el cumplimiento de las normativas y los estándares del sector.

Este AWS producto o servicio sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la [página de documentación sobre la seguridad del AWS servicio](#) y [AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad](#).

Resiliencia de este AWS producto o servicio

La infraestructura AWS global se basa en Regiones de AWS zonas de disponibilidad.

Regiones de AWS proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia.

Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de uno o varios centros de datos.

[Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte Infraestructura global.AWS](#)

Este AWS producto o servicio sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la [página de documentación sobre la seguridad del AWS servicio](#) y [AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad](#).

Seguridad de la infraestructura para este AWS producto o servicio

Este AWS producto o servicio utiliza servicios gestionados y, por lo tanto, está protegido por la seguridad de la red AWS global. Para obtener información sobre los servicios AWS de seguridad y cómo se AWS protege la infraestructura, consulte [Seguridad AWS en la nube](#). Para diseñar su AWS entorno utilizando las mejores prácticas de seguridad de la infraestructura, consulte [Protección de infraestructuras en un marco](#) de buena AWS arquitectura basado en el pilar de la seguridad.

Utiliza las llamadas a la API AWS publicadas para acceder a este AWS producto o servicio a través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad de seguridad de IAM principal. También puede utilizar [AWS Security Token Service](#) (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Este AWS producto o servicio sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la [página de documentación sobre la seguridad del AWS servicio](#) y [AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad](#).

Aplicar una versión mínima de TLS

Para aumentar la seguridad a la hora de comunicarse con AWS los servicios, configúrelos AWS SDK for JavaScript para que utilicen TLS 1.2 o una versión posterior.

Important

La AWS SDK for JavaScript versión 3 negocia automáticamente la versión de TLS de nivel más alto compatible con un punto final de servicio determinado. AWS Si lo desea, puede aplicar una versión mínima de TLS que exija su aplicación, como TLS 1.2 o 1.3, pero tenga en cuenta que algunos puntos de enlace del AWS servicio no admiten TLS 1.3, por lo que algunas llamadas podrían fallar si aplica TLS 1.3.

Transport Layer Security (TLS) es un protocolo que utilizan los navegadores web y otras aplicaciones para garantizar la privacidad e integridad de los datos intercambiados a través de una red.

Verificar y aplicar TLS en Node.js

Cuando se utiliza AWS SDK for JavaScript con Node.js, se utiliza la capa de seguridad Node.js subyacente para configurar la versión de TLS.

Node.js 12.0.0 y las versiones posteriores utilizan una versión mínima de OpenSSL 1.1.1, que admite TLS 1.3. La AWS SDK for JavaScript versión 3 usa TLS 1.3 de forma predeterminada cuando está disponible, pero usa una versión inferior si es necesaria.

Verificar la versión de OpenSSL y TLS

Para obtener la versión de OpenSSL que usa Node.js en su equipo, ejecute el siguiente comando.

```
node -p process.versions
```

La versión de OpenSSL de la lista es la versión que utiliza Node.js, como se muestra en el siguiente ejemplo.

```
openssl: '1.1.1b'
```

Para obtener la versión de TLS que usa Node.js en su equipo, inicie el shell de Node y ejecute los siguientes comandos, en orden.

```
> var tls = require("tls");  
> var tlsSocket = new tls.TLSSocket();  
> tlsSocket.getProtocol();
```

El último comando genera la versión de TLS, como se muestra en el siguiente ejemplo.

```
'TLSv1.3'
```

Node.js utiliza de forma predeterminada esta versión de TLS e intenta negociar otra versión de TLS si una llamada no se realiza correctamente.

Aplicar una versión mínima de TLS

Node.js negocia una versión de TLS cuando se produce un error en una llamada. Puede aplicar la versión de TLS mínima permitida durante esta negociación, ya sea al ejecutar un script desde la línea de comandos o por solicitud del código. JavaScript

Para especificar la versión mínima de TLS desde la línea de comandos, debe utilizar Node.js versión 11.0.0 o posterior. Para instalar una versión específica de Node.js, primero instale Node Version Manager (nvm) siguiendo los pasos que encontrará en [Instalación y actualización de Node Version Manager](#). A continuación, ejecute los siguientes comandos para instalar y usar una versión específica de Node.js.


```
nvm install 11
nvm use 11
```

Enforce TLS 1.2

Para forzar que TLS 1.2 sea la versión mínima permitida, especifique el argumento `--tls-min-v1.2` al ejecutar el script, como se muestra en el siguiente ejemplo.

```
node --tls-min-v1.2 yourScript.js
```

Para especificar la versión de TLS mínima permitida para una solicitud específica en el JavaScript código, utilice el `httpOptions` parámetro para especificar el protocolo, como se muestra en el siguiente ejemplo.

```
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 forzar que TLS 1.3 sea la versión mínima permitida, especifique el argumento `--tls-min-v1.3` al ejecutar el script, como se muestra en el siguiente ejemplo.

```
node --tls-min-v1.3 yourScript.js
```

Para especificar la versión de TLS mínima permitida para una solicitud específica en el JavaScript código, usa el `httpOptions` parámetro para especificar el protocolo, como se muestra en el siguiente ejemplo.

```
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 y aplicar TLS en un script de navegador

Cuando utilizas el SDK JavaScript en un script del navegador, la configuración del navegador controla la versión de TLS que se utiliza. La versión de TLS que utiliza el navegador no se puede detectar ni establecer mediante script y el usuario debe configurarla. Para verificar y aplicar la versión de TLS que se utiliza en un script de navegador, consulte las instrucciones de su navegador específico.

Microsoft Internet Explorer

1. Abra Internet Explorer.
2. En la barra de menús, seleccione la pestaña Herramientas ▶ Opciones de Internet ▶ Avanzadas.
3. Desplácese hacia abajo hasta la categoría Seguridad y marque manualmente la casilla de opción Usar TLS 1.2.
4. Haga clic en OK (Aceptar).
5. Cierre el navegador y reinicie Internet Explorer.

Microsoft Edge

1. En el cuadro de búsqueda del menú de Windows, escriba *Opciones de Internet*.
2. En Mejor coincidencia, haga clic en Opciones de Internet.

3. En la ventana Propiedades de Internet, en la pestaña Avanzadas, desplácese hacia abajo hasta la sección Seguridad.
4. Marque la casilla de verificación Usar TLS 1.2.
5. Haga clic en OK (Aceptar).

Google Chrome

1. Abra Google Chrome.
2. Haga clic en Alt F y seleccione Configuración.
3. Desplácese hacia abajo y seleccione Mostrar configuración avanzada...
4. Desplácese hasta la sección Sistema y haga clic en Abrir configuración de proxy...
5. Seleccione la pestaña Avanzado.
6. Desplácese hacia abajo hasta la categoría Seguridad y marque manualmente la casilla de opción Usar TLS 1.2.
7. Haga clic en OK (Aceptar).
8. Cierre el navegador y reinicie Google Chrome.

Mozilla Firefox

1. Abra Firefox.
2. En la barra de direcciones, escriba about:config y pulse Intro.
3. En el campo Buscar, escriba tls. Busque la entrada security.tls.version.min y haga doble clic en ella.
4. Establezca el valor entero en 3 para hacer que el protocolo de TLS 1.2 sea el predeterminado.
5. Haga clic en OK (Aceptar).
6. Cierre el navegador y reinicie Mozilla Firefox.

Apple Safari

No hay opciones para activar protocolos SSL. Si utiliza la versión 7 o superior de Safari, TLS 1.2 se habilita automáticamente.

Migre de la versión 2.x a la 3.x del AWS SDK for JavaScript

La AWS SDK for JavaScript versión 3 es una reescritura importante de la versión 2. En esta sección se describen las diferencias entre las dos versiones y se explica cómo migrar de la versión 2 a la versión 3 del SDK para JavaScript.

Migre su código al SDK para la JavaScript versión 3 mediante codemod

AWS SDK for JavaScript la versión 3 (v3) incluye interfaces modernizadas para las configuraciones y utilidades de los clientes, que incluyen credenciales, carga multiparte de Amazon S3, cliente de documentos DynamoDB, camareros y más. [Encontrará los cambios en la versión 2 y los equivalentes de cada cambio en la versión 3 en la guía de migración del repositorio. AWS SDK for JavaScript GitHub](#)

Para aprovechar al máximo las ventajas de la versión AWS SDK for JavaScript 3, te recomendamos que utilices los scripts de codemod que se describen a continuación.

Usa codemod para migrar el código de la versión 2 existente

La colección de scripts de codemod de [aws-sdk-js-codemod](#) le ayuda a migrar su aplicación (v2) existente para usar las API de la versión 3. AWS SDK for JavaScript Puede ejecutar la transformación de la siguiente manera.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

Por ejemplo, supongamos que tiene el siguiente código, que crea un cliente Amazon DynamoDB a partir de la versión 2 y llama a la operación `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);
});
```

```
});
```

Puede ejecutar nuestra transformación `v2-to-v3` en `example.ts` de la siguiente manera.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

La transformación convertirá la importación de DynamoDB a la versión 3, creará el cliente de la versión 3 y llamará a la operación `listTables` de la siguiente manera.

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

Hemos implementado transformaciones para casos de uso comunes. Si su código no se transforma correctamente, cree un [informe de errores](#) o una [solicitud de función](#) con un ejemplo de código de entrada y código de salida observado/esperado. Si su caso de uso específico ya está incluido en [un problema existente](#), muestre su apoyo votando a favor.

¿Qué novedades incluye la versión 3?

La versión 3 del SDK para (v3) contiene las siguientes funciones nuevas. JavaScript

Paquetes modularizados

Los usuarios ahora pueden usar un paquete independiente para cada servicio.

Nueva pila de middleware

Los usuarios ahora pueden usar una pila de middleware para controlar el ciclo de vida de una llamada a una operación.

Además, el SDK viene incorporado TypeScript, lo que tiene muchas ventajas, como la escritura estática.

Important

Los ejemplos de código para la versión 3 de esta guía están escritos en ECMAScript 6 (ES6). ES6 ofrece una nueva sintaxis y nuevas funciones para hacer que su código sea más moderno y legible, y le permite hacer más. ES6 requiere que utilice la versión 13.x o superior de Node.js. Para descargar e instalar la versión más reciente de Node.js, consulte [Descargas de Node.js](#). Para obtener más información, consulte [Sintaxis ES6/commonJS de JavaScript](#).

Paquetes modularizados

La versión 2 del SDK para la versión 2 JavaScript (v2) requería el uso del AWS SDK completo, tal y como se indica a continuación.

```
var AWS = require("aws-sdk");
```

Cargar el SDK completo no es un problema si la aplicación utiliza muchos AWS servicios. Sin embargo, si solo necesitas usar unos pocos AWS servicios, tendrás que aumentar el tamaño de la aplicación con código que no necesites o no utilices.

En la versión 3, puedes cargar y usar solo los AWS servicios individuales que necesites. Esto se muestra en el siguiente ejemplo, que le da acceso a Amazon DynamoDB (DynamoDB).

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

No solo puede cargar y usar AWS servicios individuales, sino que también puede cargar y usar solo los comandos de servicio que necesite. Esto se muestra en los siguientes ejemplos, que proporcionan acceso al cliente de DynamoDB y al comando `ListTablesCommand`.

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

Important

No debe importar submódulos a módulos. Por ejemplo, el siguiente código podría dar lugar a errores.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/  
CognitoIdentity";
```

El siguiente es el código correcto.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
```

Comparación del tamaño del código

En la versión 2 (v2), un ejemplo de código sencillo que muestre todas las tablas de Amazon DynamoDB de us-west-2 la región podría tener el siguiente aspecto.

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

La versión 3 tiene el siguiente aspecto.

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

El paquete `aws-sdk` añade unos 40 MB a la aplicación. Sustituir `var AWS = require("aws-sdk")` por `import {DynamoDB} from "@aws-sdk/client-dynamodb"` reduce esa sobrecarga a unos 3 MB. Al restringir la importación únicamente al cliente y al comando `ListTablesCommand` de DynamoDB, se reduce la sobrecarga a 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({});
```

Comandos de llamada en la v3

Puede realizar operaciones en la versión 3 mediante los comandos de la versión 2 o la versión 3. Para utilizar los comandos de la versión 3, importe los comandos y los clientes del paquete de AWS servicios necesarios y ejecute el comando mediante el `.send` método que utilice el patrón `async/await`.

Para utilizar los comandos de la versión 2, se importan los paquetes de AWS servicios necesarios y se ejecuta el comando de la versión 2 directamente en el paquete mediante un patrón de devolución de llamada o `async/await`.

Uso de los comandos de la versión 3

La versión 3 proporciona un conjunto de comandos para cada paquete de AWS servicios que le permiten realizar operaciones para ese AWS servicio. Después de instalar un servicio AWS, puede explorar los comandos disponibles en el `node-modules/@aws-sdk/client-PACKAGE_NAME/commands` folder de su proyecto.

Tiene que importar los comandos que desee utilizar. Por ejemplo, el código siguiente carga el servicio de DynamoDB y el comando `CreateTableCommand`.


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

Para llamar a estos comandos con el patrón `async/await` recomendado, utilice la sintaxis siguiente.

```
CLIENT.send(new XXXCommand);
```

Por ejemplo, en el ejemplo siguiente se crea una tabla de DynamoDB con el patrón `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);
  }
}) ();
```

Uso de comandos de la versión 2

Para usar los comandos de la versión 2 en el SDK JavaScript, se importan los paquetes de AWS servicio completos, como se muestra en el código siguiente.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

Para llamar a los comandos de la versión 2 con el patrón `asíncrono/de espera` recomendado, usa la siguiente sintaxis.

```
client.command(parameters);
```

En el siguiente ejemplo, se utiliza el `createTable` comando v2 para crear una tabla de DynamoDB con el patrón `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();
```

En el siguiente ejemplo, se utiliza el `createBucket` comando v2 para crear un bucket de Amazon S3 mediante el patrón `callback`.

```
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();
```

Nueva pila de middleware

La versión 2 del SDK le permitía modificar una solicitud a lo largo de las múltiples etapas de su ciclo de vida al adjuntar detectores de eventos a la solicitud. Este enfoque puede dificultar la depuración de lo que salió mal durante el ciclo de vida de una solicitud.

En la versión 3, puedes usar una nueva pila de middleware para controlar el ciclo de vida de una llamada a una operación. Este enfoque ofrece un par de ventajas. Cada etapa de middleware de

la pila llama a la siguiente etapa de middleware después de realizar cualquier cambio en el objeto de solicitud. Esto también facilita mucho la depuración de problemas en la pila, ya que permite ver exactamente a qué etapas del middleware se llamó antes de que se produjera el error.

En el siguiente ejemplo, se añade un encabezado personalizado a un cliente de Amazon DynamoDB (que hemos creado y mostrado anteriormente) mediante middleware. El primer argumento es una función que acepta `next`, que es la siguiente fase de middleware de la pila a la que se debe llamar, y `context`, que es un objeto que contiene información sobre la operación a la que se está llamando. La función devuelve una función que acepta `args`, que es un objeto que contiene los parámetros pasados a la operación y a la solicitud. Devuelve el resultado de llamar al siguiente middleware con `args`.

```
dbclient.middlewareStack.add(
  (next, context) => args => {
    args.request.headers["Custom-Header"] = "value";
    return next(args);
  },
  {
    step: "build"
  }
);

dbclient.send(new PutObjectCommand(params));
```

¿Qué hay de diferente entre la AWS SDK for JavaScript v2 y la v3?

En esta sección se muestran los cambios notables de la AWS SDK for JavaScript v2 a la v3. Como la v3 es una reescritura modular de la v2, algunos conceptos básicos son diferentes entre la v2 y la v3. [Puedes obtener más información sobre estos cambios en las publicaciones de nuestro blog](#). Las siguientes publicaciones de blog te ayudarán a ponerte al día:

- [Paquetes modulares en AWS SDK for JavaScript](#)
- [Presentamos Middleware Stack in Modular AWS SDK for JavaScript](#)

A continuación se presenta un resumen de los cambios en la AWS SDK for JavaScript interfaz de la versión 2 a la versión 3. El objetivo es ayudarle a encontrar fácilmente los equivalentes de la versión 3 de las API de la versión 2 con las que ya está familiarizado.

Constructores de clientes

Esta lista está indexada según los parámetros de [configuración de la versión 2](#).

- [computeChecksums](#)
 - v2: si se deben calcular las sumas de control de MD5 para los cuerpos de carga útil cuando el servicio las acepte (actualmente solo se admiten en S3).
 - v3: los comandos aplicables de S3 (PutObject, PutBucketCors, etc.) calcularán automáticamente las sumas de comprobación MD5 de la carga útil solicitada. También puede especificar un algoritmo de suma de comprobación diferente en el `ChecksumAlgorithm` parámetro de los comandos para utilizar un algoritmo de suma de comprobación diferente. [Puede encontrar más información en el anuncio de la función S3](#)
- [convertResponseTypes](#)
 - v2: Si los tipos se convierten al analizar los datos de respuesta.
 - v3: En desuso. Se considera que esta opción no es segura para escribir porque no convierte los tipos, como la marca de tiempo o los binarios en base64, de la respuesta JSON.
- [correctClockSkew](#)
 - v2: si se debe aplicar una corrección de la distorsión del reloj y volver a intentar las solicitudes que no se puedan procesar debido a una distorsión del reloj del cliente.
 - v3: Obsoleto. El SDK siempre aplica una corrección de la inclinación del reloj.
- [systemClockOffset](#)
 - v2: un valor de compensación en milisegundos que se aplica a todos los tiempos de firma.
 - v3: Sin cambios.
- [credentials](#)
 - v2: Las AWS credenciales con las que se firman las solicitudes.
 - v3: Sin cambios. También puede ser una función asíncrona que devuelve credenciales. Si la función devuelve un `expiration` (`Date`), se volverá a llamar a la función cuando se acerque la fecha y hora de caducidad. Consulte la referencia de la [versión 3 para](#) ver las credenciales. `AwsAuthInputConfig`
- [endpointCacheSize](#)
 - v2: el tamaño de la caché global que almacena los puntos finales de las operaciones de descubrimiento de puntos finales.
 - v3: Sin cambios.
- [endpointDiscoveryEnabled](#)
 - v2: Indica si se deben llamar a las operaciones con puntos finales proporcionados por el servicio de forma dinámica.

- v3: Sin cambios.
- [hostPrefixEnabled](#)
 - v2: si se deben agrupar los parámetros de la solicitud con el prefijo del nombre de host.
 - v3: Obsoleto. El SDK siempre inyecta el prefijo del nombre de host cuando es necesario.
- [httpOptions](#)

Un conjunto de opciones para pasarlas a la solicitud HTTP de bajo nivel. Estas opciones se agregan de forma diferente en la versión 3. Puede configurarlas proporcionando una `requestHandler`. Este es el ejemplo de cómo configurar las opciones http en el tiempo de ejecución de Node.js. Puede encontrar más información en la [referencia de la versión 3 para NodeHttpHandler](#).

Todas las solicitudes de la versión 3 utilizan HTTPS de forma predeterminada. Solo necesitas proporcionar un `HttpAgent` 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*/
  }),
});
```

Si está pasando un punto final personalizado que usa http, debe proporcionar `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",
});
```

```
});
```

Si el cliente se ejecuta en navegadores, hay disponible un conjunto diferente de opciones. Puede encontrar más información en la [referencia de la versión 3 para FetchHttpHandler](#).

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
const dynamodbClient = new DynamoDBClient({
  requestHandler: new FetchHttpHandler({
    requestTimeout: /* number in milliseconds */
  }),
});
```

Cada opción de `httpOptions` se especifica a continuación:

- `proxy`
 - v2: La URL a través de la cual se envían las solicitudes de proxy
 - v3: Puede configurar un proxy con un agente siguiendo los pasos de [Configuración de proxies para Node.js](#)
- `agent`
 - v2: El objeto del agente con el que se realizan las solicitudes HTTP. Se utiliza para agrupar conexiones.
 - v3: Puede configurarlo `httpAgent` o `httpsAgent` como se muestra en los ejemplos anteriores.
- `connectTimeout`
 - v2: Establece el tiempo de espera del socket después de no poder establecer una conexión con el servidor después de milisegundos de `ConnectTimeout`.
 - v3: [connectionTimeout](#) está disponible en opciones. `NodeHttpHandler`
- `timeout`
 - v2: el número de milisegundos que puede tardar una solicitud en finalizar automáticamente.
 - v3: `socketTimeout` está disponible [en NodeHttpHandler opciones](#).
- `xhrAsync`
 - v2: Si el SDK enviará solicitudes HTTP asíncronas.
 - v3: En desuso. Las solicitudes son siempre asíncronas.
- `xhrWithCredentials`
 - v2: Establece la propiedad «withCredentials» de un objeto XML. `HttpRequest`
 - v3: No disponible. El SDK hereda [las configuraciones de recuperación predeterminadas](#)

- v2: un objeto que responde a `.write ()` (como una transmisión) o `.log ()` (como el objeto de la consola) para registrar información sobre las solicitudes.
- v3: Sin cambios. En la versión 3 hay registros más detallados disponibles.
- [maxRedirects](#)
 - v2: la cantidad máxima de redireccionamientos que se deben seguir para una solicitud de servicio.
 - v3: Obsoleto. El SDK no sigue los redireccionamientos para evitar solicitudes involuntarias entre regiones.
- [maxRetries](#)
 - v2: la cantidad máxima de reintentos que se deben realizar para una solicitud de servicio.
 - v3: Cambiado a `maxAttempts`. Consulte más información en la [referencia de la versión 3](#) para `RetryInputConfig`. Tenga en cuenta que `maxAttempt` debería ser `maxRetries + 1`.
- [paramValidation](#)
 - v2: Si los parámetros de entrada deben validarse con la descripción de la operación antes de enviar la solicitud.
 - v3: Obsoleto. El SDK no realiza la validación en el lado del cliente durante el tiempo de ejecución.
- [region](#)
 - v2: la región a la que se van a enviar las solicitudes de servicio.
 - v3: Sin cambios. También puede ser una función asíncrona que devuelve una cadena de región.
- [retryDelayOptions](#)
 - v2: conjunto de opciones para configurar el retraso del reintento en caso de errores que se puedan volver a intentar.
 - v3: En desuso. El SDK admite una estrategia de reintento más flexible con la opción de constructor de `retryStrategy` clientes. Consulte más información [en la referencia de la versión 3](#).
- [s3BucketEndpoint](#)
 - v2: si el punto final proporcionado se dirige a un depósito individual (falso si se dirige al punto final de la API raíz).
 - v3: Cambiado a `bucketEndpoint`. Consulte más información en la [referencia de la versión 3 para BucketEndpoint](#). Tenga en cuenta que si se establece en `true`, si se especifica el punto final de la solicitud en el parámetro de `Bucket` solicitud, se sobrescribirá el punto final original. Mientras que en la versión 2, el punto final de la solicitud en el constructor del cliente sobrescribe el parámetro de la `Bucket` solicitud.
- [s3DisableBodySigning](#)

- v2: si se debe deshabilitar la firma corporal de S3 al usar la versión de firma v4.
- v3: Se cambió el nombre a `applyChecksum`
- [s3ForcePathStyle](#)
 - v2: Si se deben forzar las URL con estilos de ruta para los objetos de S3.
 - v3: Se ha cambiado el nombre a `forcePathStyle`
- [s3UseArnRegion](#)
 - v2: si se debe anular la región de solicitud con la región inferida del ARN del recurso solicitado.
 - v3: se cambió el nombre a `useArnRegion`
- [s3UseEast1RegionalEndpoint](#)
 - v2: cuando la región está configurada como «us-east-1», ya sea para enviar una solicitud s3 a puntos finales globales o puntos finales regionales «us-east-1».
 - v3: Obsoleto. El cliente S3 siempre utilizará el punto final regional si la región está configurada en. `us-east-1` Puede configurar la región para `aws-global` enviar las solicitudes al punto final global de S3.
- [signatureCache](#)
 - v2: Si la firma con la que se firman las solicitudes (anulando la configuración de la API) está en caché.
 - v3: En desuso. El SDK siempre guarda en caché las claves de firma cifradas.
- [signatureVersion](#)
 - v2: la versión de firma con la que se firman las solicitudes (invalida la configuración de la API).
 - v3: Obsoleta. La firma V2, compatible con el SDK de la versión 2, ha quedado obsoleta AWS. La versión 3 solo admite la firma v4.
- [sslEnabled](#)
 - v2: Si el SSL está habilitado para las solicitudes.
 - v3: Se cambió el nombre a. `tls`
- [stsRegionalEndpoints](#)
 - v2: Si se debe enviar la solicitud sts a puntos finales globales o regionales.
 - v3: En desuso. El cliente STS siempre utilizará puntos finales regionales si se establece en una región específica. Puede configurar la región para enviar la solicitud `aws-global` al punto final global de STS.
- [useAccelerateEndpoint](#)
 - v2: si se debe utilizar el punto final Accelerate con el servicio S3.
 - v3: Sin cambios.

Proveedores de credencial

En la versión 2, el SDK JavaScript proporciona una lista de proveedores de credenciales entre los que elegir, así como una cadena de proveedores de credenciales, disponible de forma predeterminada en Node.js, que intenta cargar las AWS credenciales de todos los proveedores más comunes. El SDK para la JavaScript versión 3 simplifica la interfaz del proveedor de credenciales, lo que facilita el uso y la creación de proveedores de credenciales personalizados. Además de una nueva cadena de proveedores de credenciales, el SDK de la versión JavaScript 3 incluye una lista de proveedores de credenciales que pretenden ofrecer una versión equivalente a la versión 2.

Estos son todos los proveedores de credenciales de la versión 2 y sus equivalentes de la versión 3.

Proveedor de credenciales predeterminado

El proveedor de credenciales predeterminado es la forma en que el SDK JavaScript resuelve la AWS credencial si no se proporciona una de forma explícita.

- v2: [CredentialProviderChain in Node.js](#) resuelve las credenciales de las fuentes en el siguiente orden:
 - [Variable ambiental](#)
 - [Archivo de credenciales compartido](#)
 - [Credenciales del contenedor ECS](#)
 - [Proceso externo de generación](#)
 - [Token OIDC del archivo especificado](#)
 - [Metadatos de la instancia EC2](#)

Si uno de los proveedores de credenciales anteriores no resuelve la AWS credencial, la cadena pasa al siguiente proveedor hasta que se resuelva una credencial válida, y la cadena generará un error cuando todos los proveedores fallen.

En los tiempos de ejecución de Browser y React Native, la cadena de credenciales está vacía y las credenciales deben configurarse de forma explícita.

- v3: [DefaultProvider](#). Las fuentes de credenciales y el orden alternativo no cambian en la versión 3. [También admite credenciales.AWS IAM Identity Center](#)

Credenciales temporales

- v2: [ChainableTemporaryCredentials](#) representa las credenciales temporales recuperadas de `AWS.STS`. Sin ningún parámetro adicional, las credenciales se obtendrán de la `AWS.STS.getSessionToken()` operación. Si se proporciona una función de IAM, la `AWS.STS.assumeRole()` operación se utilizará en su lugar para obtener las credenciales de la función. `AWS.ChainableTemporaryCredentials` difiere de la forma `AWS.TemporaryCredentials` en que se gestionan las `MasterCredentials` y las actualizaciones. `AWS.ChainableTemporaryCredentials` actualiza las credenciales caducadas utilizando las `MasterCredentials` transferidas por el usuario para permitir el encadenamiento de las credenciales de STS. Sin embargo, contrae las `MasterCredentials` de `AWS.TemporaryCredentials` forma recursiva durante la instanciación, lo que impide actualizar las credenciales que requieren credenciales intermedias y temporales.

El original [TemporaryCredentials](#) ha quedado obsoleto en favor de la versión 2. `ChainableTemporaryCredentials`

- v3: [fromTemporaryCredentials](#) Puedes llamar `fromTemporaryCredentials()` desde el `@aws-sdk/credential-providers` paquete. A continuación se muestra un ejemplo:

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

Credenciales de identidad de Amazon Cognito

Cargue las credenciales del servicio Amazon Cognito Identity, que se utiliza normalmente en los navegadores.

- v2: [CognitoIdentityCredentials](#) Representa las credenciales recuperadas de STS Web Identity Federation mediante el servicio Amazon Cognito Identity.
- v3: [Cognito Identity Credential Provider](#) El [@aws/credential-providers](#) paquete proporciona dos funciones de proveedor de credenciales, una de las cuales [fromCognitoIdentity](#) toma un identificador de identidad y realiza llamadas `cognitoIdentity:GetCredentialsForIdentity`, mientras que la otra utiliza un identificador de grupo de identidades, [fromCognitoIdentityPool](#) realiza la primera invocación y, a continuación, llama `cognitoIdentity:GetId`. `fromCognitoIdentity` Las invocaciones posteriores de esta última no se vuelven a invocar. `GetId`

El proveedor implementa el «flujo simplificado» que se describe en la Guía [para desarrolladores de Amazon Cognito](#). No se admite el «flujo clásico», que implica llamar `cognito:GetOpenIdToken` y luego llamar `sts:AssumeRoleWithWebIdentity`. Envíanos una [solicitud de función](#) si la necesitas.

```
// 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 metadatos de EC2 (IMDS)

Representa las credenciales recibidas del servicio de metadatos en una instancia de Amazon EC2.

- v2: [EC2MetadataCredentials](#)
- v3: [fromInstanceMetadata](#): Crea un proveedor de credenciales que obtendrá las credenciales del servicio de metadatos de instancias Amazon EC2.

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

Credenciales ECS

Representa las credenciales recibidas de la URL especificada. Este proveedor solicitará credenciales temporales del URI especificado por la variable de entorno `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` o por la variable de entorno `AWS_CONTAINER_CREDENTIALS_FULL_URI`.

- v2: `ECSCredentials` o [RemoteCredentials](#).
- v3: [fromContainerMetadata](#) crea un proveedor de credenciales que obtendrá las credenciales del 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
  }),
});
```

Credenciales del sistema de archivos

- v2: [FileSystemCredentials](#) representa las credenciales de un archivo JSON del disco.
- v3: Obsoleto. Puede leer explícitamente el archivo JSON y proporcionárselo al cliente. Si la necesitas, envíanos una [solicitud de función](#).

Proveedor de credenciales SAML

- v2: [SAMLCredentials](#) representa las credenciales recuperadas del soporte SAML de STS.
- v3: No disponible. Por favor, envíenos una [solicitud de función](#) si la necesita.

Credenciales del archivo de credenciales compartido

Carga las credenciales del archivo de credenciales compartido (predeterminado `~/.aws/credentials` o definido por la variable de entorno `AWS_SHARED_CREDENTIALS_FILE`). Este archivo es compatible con diferentes AWS SDK y herramientas. Puedes consultar el [documento de archivos de configuración y credenciales compartidos](#) para obtener más información.

- 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
  }),
});
```

Credenciales de identidad web

Recupera las credenciales mediante el token OIDC de un archivo en el disco. Se usa habitualmente en 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 },
  }),
});
```

```
});
```

Credenciales de federación de identidades web

Recupera las credenciales del soporte de federación de identidades web de 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 },
  }),
});
```

Consideraciones sobre Amazon S3

Carga multiparte en Amazon S3

En la versión 2, el cliente Amazon S3 contiene una [upload\(\)](#) operación que permite cargar objetos grandes con la [función de carga multiparte que ofrece Amazon S3](#).

En la versión 3, el [@aws-sdk/lib-storage](#) paquete está disponible. Es compatible con todas las funciones que se ofrecen en la versión 2 `upload()` y es compatible con Node.js y con el tiempo de ejecución de los navegadores.

URL prefirmada de Amazon S3

En la versión 2, el cliente Amazon S3 contiene las [getSignedUrlPromise\(\)](#) operaciones [getSignedUrl\(\)](#) y para generar una URL que los usuarios pueden utilizar para cargar o descargar objetos de Amazon S3.

En la versión 3, el [@aws-sdk/s3-request-presigner](#) paquete está disponible. Este paquete contiene las funciones tanto para las `getSignedUrlPromise()` operaciones como para las `getSignedUrl()` mismas. Esta [entrada de blog](#) analiza los detalles de este paquete.

Redireccionamientos de región de Amazon S3

Si se transfiere una región incorrecta al cliente de Amazon S3 y se produce un error posterior `PermanentRedirect` (estado 301), el cliente Amazon S3 de la versión 3 admite los redireccionamientos de región (anteriormente denominados Amazon S3 Global Client en la versión 2). Puede usar el [followRegionRedirects](#) indicador en la configuración del cliente para hacer que el cliente de Amazon S3 siga las redirecciones regionales y respalde su función como cliente global.

Note

Tenga en cuenta que esta función puede generar una latencia adicional, ya que las solicitudes fallidas se reintentan con una región corregida cuando se recibe un `PermanentRedirect` error con el estado 301. Esta función solo debe usarse si no conoces con antelación la región de tus depósitos.

Historial de documentos de AWS SDK for JavaScript la versión 3

Historial de documentos

En la siguiente tabla se describen los cambios importantes con el lanzamiento de la V3 de AWS SDK for JavaScript a partir del 20 de octubre de 2020. Para obtener notificaciones sobre las actualizaciones de esta documentación, puede suscribirse a una [fuente RSS](#).

Cambio	Descripción	Fecha
Anuncio	Se actualizó el banner superior con un end-of-support recordatorio para Internet Explorer 11.	23 de septiembre de 2022
Actualizaciones menores	Actualizaciones menores para mejorar la claridad y resolver los enlaces rotos. Se han añadido enlaces de información a la guía de referencia de AWS los SDK y las herramientas.	22 de agosto de 2022
Imponga una versión mínima de TLS	Se ha agregado información sobre TLS 1.3.	31 de marzo de 2022
Tutorial actualizado AWS Lambda	Se ha añadido un tutorial que muestra cómo crear una aplicación basada en un navegador para enviar datos a una tabla de Amazon DynamoDB.	20 de octubre de 2020
Se actualizó el tema Establecer credenciales en Node.js	Tema actualizado sobre la configuración de credenciales	20 de octubre de 2020

	en Node.js para la AWS SDK for JavaScript versión 3.	
Migre a la versión 3	Se agregó un tema para describir cómo migrar a la AWS SDK for JavaScript v3.	20 de octubre de 2020
Cómo empezar	Se han actualizado los temas para empezar a usar el navegador y empezar a utilizar Node.js para la AWS SDK for JavaScript versión 3.	20 de octubre de 2020
Browser Builder	Se ha eliminado la información sobre AWS Browser Builder porque no es necesaria para la AWS SDK for JavaScript versión 3.	20 de octubre de 2020
Ejemplos del servicio Amazon Transcribe actualizados	Ejemplos actualizados del servicio Amazon Transcribe para AWS SDK for JavaScript la versión 3.	20 de octubre de 2020
Se han actualizado los ejemplos de Amazon Simple Notification Service	Ejemplos actualizados del servicio Amazon Simple Notification Service para la AWS SDK for JavaScript versión 3.	20 de octubre de 2020
Se han actualizado los ejemplos de Amazon Simple Email Service	Ejemplos actualizados del servicio Amazon Simple Email Service para la AWS SDK for JavaScript versión 3.	20 de octubre de 2020

Se han actualizado los ejemplos del servicio Amazon Redshift	Se han actualizado los ejemplos del servicio Amazon Redshift para AWS SDK for JavaScript la versión 3.	20 de octubre de 2020
Se han actualizado los ejemplos del servicio Amazon Lex	Ejemplos actualizados del servicio Amazon Lex para la AWS SDK for JavaScript versión 3.	20 de octubre de 2020
Se han actualizado los ejemplos del servicio Amazon DynamoDB	Se han actualizado los ejemplos del servicio Amazon DynamoDB para la versión 3. AWS SDK for JavaScript	20 de octubre de 2020
AWS Elemental MediaConvert ejemplos de servicios actualizados	Ejemplos AWS Elemental MediaConvert de servicios actualizados para la AWS SDK for JavaScript versión 3.	20 de octubre de 2020
AWS Lambda ejemplos de servicios actualizados	Ejemplos AWS Lambda de servicios actualizados para la AWS SDK for JavaScript versión 3.	20 de octubre de 2020
AWS SDK for JavaScript Vista previa de la guía para desarrolladores de V3	Publicada la versión preliminar de la Guía para desarrolladores de la AWS SDK for JavaScript V3.	19 de octubre de 2020