Guida per gli sviluppatori per la versione SDK 3

AWS SDK for JavaScript

AWS SDK for JavaScript: Guida per gli sviluppatori per la versione SDK 3

Table of Contents

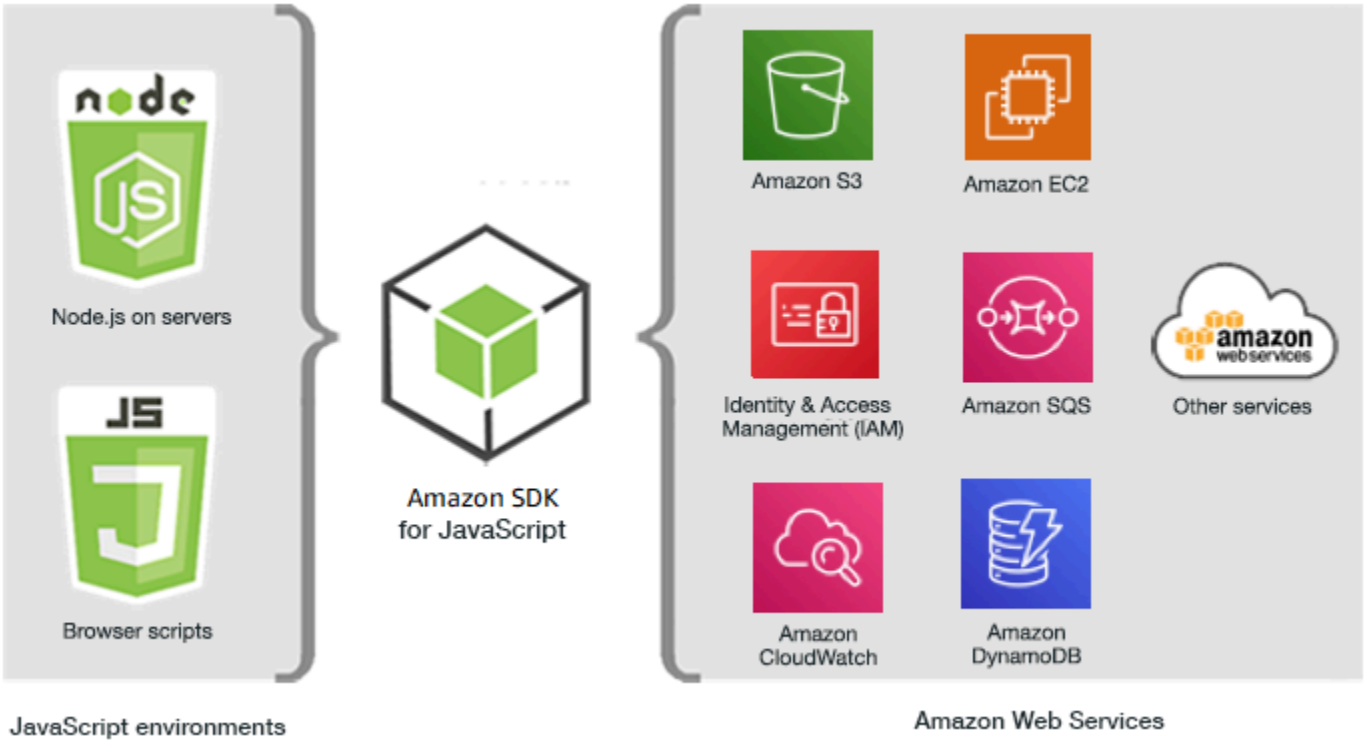
La Guida di riferimento dell'API AWS SDK for JavaScript V3 descrive in dettaglio tutte le operazioni API per la AWS SDK for JavaScript versione 3 (V3).

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.

Qual è ilAWS SDK for JavaScript?

Benvenuto nella guida per sviluppatori di AWS SDK for JavaScript. Questa guida fornisce informazioni generali sulla configurazione e l'impostazione di. AWS SDK for JavaScript Inoltre, illustra esempi e tutorial sull'AWS SDK for JavaScriptesecuzione di vari AWS servizi utilizzando.

La Guida di riferimento alle API AWS SDK for JavaScript v3 fornisce un' JavaScript API per AWS i servizi. È possibile utilizzare l' JavaScript API per creare librerie o applicazioni per Node.js o per il browser.



Manutenzione e supporto per le versioni principali dell'SDK

Per informazioni sulla manutenzione e sul supporto per le versioni principali dell'SDK e le relative dipendenze sottostanti, consulta quanto segue nella Guida di riferimento degli strumenti e degli SDK AWS:

- Policy di manutenzione degli SDK AWS e degli strumenti
- Matrice di supporto delle versioni degli SDK AWS e degli strumenti

Cosa c'è di nuovo nella versione 3

La versione 3 dell'SDK for JavaScript (V3) contiene le seguenti nuove funzionalità.

Pacchetti modularizzati

Gli utenti possono ora utilizzare un pacchetto separato per ogni servizio.

Nuovo stack di middleware

Gli utenti possono ora utilizzare uno stack middleware per controllare il ciclo di vita di una chiamata operativa.

Inoltre, l'SDK è integrato, il che presenta molti vantaggi TypeScript, come la digitazione statica.

> **Important**
>
> Gli esempi di codice per V3 in questa guida sono scritti in ECMAScript 6 (ES6). ES6 offre una nuova sintassi e nuove funzionalità per rendere il codice più moderno e leggibile e fare di più. ES6 richiede l'utilizzo di Node.js versione 13.x o successiva. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. Per ulteriori informazioni, consulta JavaScript Sintassi ES6/CommonJS.

Pacchetti modularizzati

La versione 2 dell'SDK for JavaScript (V2) richiedeva l'utilizzo dell'intero AWS SDK, come segue.

```
var AWS = require("aws-sdk");
```

Il caricamento dell'intero SDK non è un problema se l'applicazione utilizza molti servizi. AWS Tuttavia, se devi utilizzare solo alcuni AWS servizi, significa aumentare le dimensioni dell'applicazione con codice che non ti serve o che non usi.

In V3, puoi caricare e utilizzare solo i singoli AWS servizi di cui hai bisogno. Questo è illustrato nell'esempio seguente, che consente di accedere ad Amazon DynamoDB (DynamoDB).

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

Non solo puoi caricare e utilizzare singoli AWS servizi, ma puoi anche caricare e utilizzare solo i comandi di servizio necessari. Ciò è illustrato negli esempi seguenti, che consentono di accedere al client DynamoDB e al comando. `ListTablesCommand`

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

Important

Non è necessario importare sottomoduli nei moduli. Ad esempio, il codice seguente potrebbe causare errori.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/
CognitoIdentity";
```

Quello che segue è il codice corretto.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
```

Confronto delle dimensioni del codice

Nella versione 2 (V2), un semplice esempio di codice che elenca tutte le tabelle Amazon DynamoDB `us-west-2` nella regione potrebbe essere simile al seguente.

```
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 versione V3 ha il seguente aspetto.

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

Il `aws-sdk` pacchetto aggiunge circa 40 MB all'applicazione. La sostituzione `var AWS = require("aws-sdk")` con `import {DynamoDB} from "@aws-sdk/client-dynamodb"` riduce tale sovraccarico a circa 3 MB. Limitando l'importazione solo al client `ListTablesCommand` e al comando DynamoDB si riduce il sovraccarico a meno di 100 KB.

```
// Load the DynamoDB client and ListTablesCommand command for Node.js
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbclient = new DynamoDBClient({});
```

Chiamare i comandi in V3

È possibile eseguire operazioni in V3 utilizzando i comandi V2 o V3. Per utilizzare i comandi V3, è necessario importare i comandi e i client del pacchetto AWS Services richiesti ed eseguire il comando utilizzando il `.send` metodo che utilizza il modello async/await.

Per utilizzare i comandi V2, importate i pacchetti AWS Services richiesti ed eseguite il comando V2 direttamente nel pacchetto utilizzando un callback o un pattern async/await.

Utilizzo dei comandi V3

V3 fornisce un set di comandi per ogni pacchetto di AWS servizio per consentire all'utente di eseguire operazioni per quel AWS servizio. Dopo aver installato un AWS servizio, puoi sfogliare i comandi disponibili nel tuo progetto `node-modules/@aws-sdk/client-`*`PACKAGE_NAME`*`/commands folder.`

È necessario importare i comandi che si desidera utilizzare. Ad esempio, il codice seguente carica il servizio DynamoDB e il comando. `CreateTableCommand`

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

Per chiamare questi comandi nel modello async/await consigliato, utilizzate la seguente sintassi.

```
CLIENT.send(new XXXCommand)
```

Ad esempio, l'esempio seguente crea una tabella DynamoDB utilizzando il pattern async/await consigliato.

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

Utilizzo dei comandi V2

Per utilizzare i comandi V2 nell'SDK for JavaScript, è necessario importare i pacchetti di AWS servizio completi, come illustrato nel codice seguente.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

Per chiamare i comandi V2 nel pattern async/await consigliato, utilizzate la seguente sintassi.

```
client.command(parameters)
```

L'esempio seguente utilizza il `createTable` comando V2 per creare una tabella DynamoDB utilizzando il pattern async/await consigliato.

```
const {DynamoDB} = require('@aws-sdk/client-dynamodb');
const dymamoDB = new DynamoDB({region: 'us-west-2'});
var tableParams = {
    TableName : TABLE_NAME
};
async function run() => {
      try {
           const data = await dymamoDB.createTable(tableParams);
           console.log("Success", data);
      }
      catch (err) {
           console.log("Error", err);
      }
};
run();
```

L'esempio seguente utilizza il `createBucket` comando V2 per creare un bucket Amazon S3 utilizzando il pattern di 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);
         }
    })
};
```

Nuovo stack di middleware

La versione 2 dell'SDK consentiva di modificare una richiesta durante le diverse fasi del suo ciclo di vita collegando listener di eventi alla richiesta. Questo approccio può rendere difficile il debug di ciò che è andato storto durante il ciclo di vita di una richiesta.

In V3, è possibile utilizzare un nuovo stack middleware per controllare il ciclo di vita di una chiamata operativa. Questo approccio offre un paio di vantaggi. Ogni fase del middleware dello stack chiama la fase middleware successiva dopo aver apportato modifiche all'oggetto della richiesta. Ciò semplifica notevolmente anche i problemi di debug nello stack, poiché è possibile vedere esattamente quali fasi del middleware sono state chiamate prima dell'errore.

L'esempio seguente aggiunge un'intestazione personalizzata a un client Amazon DynamoDB (che abbiamo creato e mostrato in precedenza) utilizzando il middleware. Il primo argomento è una funzione che accetta`next`, che è la fase successiva del middleware dello stack da chiamare`context`, e che è un oggetto che contiene alcune informazioni sull'operazione chiamata. La funzione restituisce una funzione che accetta`args`, ovvero un oggetto che contiene i parametri passati all'operazione e alla richiesta. Restituisce il risultato della chiamata al middleware successivo con. `args`

```
dbclient.middlewareStack.add(
  (next, context) => args => {
    args.request.headers["Custom-Header"] = "value";
    return next(args);
  },
  {
    step: "build"
  }
);

dbclient.send(new PutObjectCommand(params));
```

Utilizzo dell'SDK con Node.js

Node.js è un runtime multipiattaforma per l'esecuzione di applicazioni lato server. JavaScript Puoi configurare Node.js su un'istanza Amazon Elastic Compute Cloud (Amazon EC2) per l'esecuzione su un server. Puoi anche utilizzare Node.js per scrivere funzioni AWS Lambda on demand.

L'utilizzo dell'SDK per Node.js è diverso dal modo in cui lo si utilizza JavaScript in un browser Web. La differenza sta nel modo in cui si carica l'SDK e in cui si ottengono le credenziali necessarie per

accedere a servizi Web specifici. Quando l'uso di particolari API differisce tra Node.js e il browser, evidenziamo tali differenze.

Utilizzo dell'SDK con AWS Cloud9

È inoltre possibile sviluppare applicazioni Node.js utilizzando l'SDK for JavaScript nell'AWS Cloud9IDE. Per ulteriori informazioni sull'utilizzo AWS Cloud9 con l'SDK per JavaScript, consulta. Utilizzare AWS Cloud9 con AWS SDK for JavaScript

Utilizzo dell'SDK con AWS Amplify

Per le app web, mobili e ibride basate su browser, puoi anche utilizzare la AWS Amplify libreria su. GitHub Estende l'SDK per JavaScript, fornendo un'interfaccia dichiarativa.

> **Note**
>
> I framework come Amplify potrebbero non offrire lo stesso supporto per i browser dell'SDK. JavaScript Consulta la documentazione del framework per i dettagli.

Utilizzo dell'SDK con i browser Web

Tutti i principali browser Web supportano l'esecuzione di. JavaScript JavaScript il codice in esecuzione in un browser Web viene spesso definito lato client JavaScript.

Per un elenco dei browser supportati dall'AWS SDK for JavaScript consulta Browser Web supportati.

L'utilizzo dell'SDK for JavaScript in un browser Web è diverso dal modo in cui lo si utilizza per Node.js. La differenza sta nel modo in cui si carica l'SDK e in cui si ottengono le credenziali necessarie per accedere a servizi Web specifici. Quando l'uso di API particolari differisce tra Node.js e il browser, evidenziamo tali differenze.

Utilizzo dei browser in V3

V3 consente di raggruppare e includere nel browser solo l'SDK per JavaScript i file necessari, riducendo il sovraccarico.

Per utilizzare la versione 3 dell'SDK for JavaScript nelle pagine HTML, è necessario raggruppare i moduli client richiesti e tutte le JavaScript funzioni richieste in un unico JavaScript file utilizzando Webpack e aggiungerlo in un tag script nelle pagine HTML. <head> Ad esempio:

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

> **Note**
>
> Per ulteriori informazioni su Webpack, consulta. Raggruppa le applicazioni con webpack

Per utilizzare la versione 2 dell'SDK per JavaScript, aggiungi invece un tag di script che rimanda alla versione più recente dell'SDK V2. Per ulteriori informazioni, consulta l'esempio nella Developer Guide v2. AWS SDK for JavaScript

Casi di utilizzo comune

L'utilizzo dell'SDK per JavaScript gli script del browser consente di realizzare una serie di casi d'uso interessanti. Di seguito sono riportate diverse idee su cosa è possibile creare in un'applicazione browser utilizzando l'SDK per accedere JavaScript a vari servizi Web.

- Crea una console personalizzata per AWS i servizi in cui puoi accedere e combinare funzionalità di diverse regioni e servizi per soddisfare al meglio le tue esigenze organizzative o di progetto.
- Usa Amazon Cognito Identity per consentire l'accesso degli utenti autenticati alle applicazioni del browser e ai siti Web, incluso l'uso dell'autenticazione di terze parti da Facebook e altri.
- Usa Amazon Kinesis per elaborare flussi di clic o altri dati di marketing in tempo reale.
- Usa Amazon DynamoDB per la persistenza dei dati senza server, ad esempio le preferenze dei singoli utenti per i visitatori del sito Web o gli utenti delle applicazioni.
- Utilizzare AWS Lambda per incapsulare la logica proprietaria che è possibile richiamare dagli script del browser senza scaricare e rivelare la proprietà intellettuale agli utenti.

Informazioni sugli esempi

Puoi cercare JavaScript esempi nell'SDK nel AWSCode Example Repository.

Risorse

Oltre a questa guida, sono disponibili le seguenti risorse online per SDK per sviluppatori: JavaScript

- AWS SDK for JavaScriptGuida di riferimento all'API V3
- AWSGuida di riferimento agli SDK e agli strumenti: contiene impostazioni, funzionalità e altri concetti fondamentali comuni tra gli SDK. AWS
- JavaScript Blog per sviluppatori
- AWS JavaScript Forum
- JavaScript esempi nel AWS Code Catalog
- AWSRepository di esempi di codice
- Canale Gitter
- Stack Overflow
- Domande Stack Overflow etichettate AWS -sdk-js
- GitHub
  - Fonte SDK
  - Fonte della documentazione

Nozioni di base su AWS SDK for JavaScript

AWS SDK for JavaScriptFornisce l'accesso ai servizi Web in un browser o in un ambiente Node.js. Questa sezione contiene esercizi introduttivi che mostrano come utilizzare l'SDK for JavaScript in ciascuno di questi JavaScript ambienti.

> **Note**
>
> È possibile sviluppare applicazioni Node.js e, JavaScript per le applicazioni basate su browser, utilizzando l'SDK for JavaScript nell'IDE. AWS Cloud9 Per un esempio di utilizzo AWS Cloud9 per lo sviluppo di Node.js, consulta. Utilizzare AWS Cloud9 con AWS SDK for JavaScript

Argomenti

Autenticazione SDK con AWS

È necessario definire la modalità di autenticazione del codice con AWS durante lo sviluppo con i Servizi AWS. Puoi configurare l'accesso programmatico alle AWS risorse in diversi modi a seconda dell'ambiente e dell'AWSaccesso a tua disposizione.

Per scegliere il metodo di autenticazione e configurarlo per l'SDK, consulta Autenticazione e accesso nella Guida di riferimento agli AWSSDK e agli strumenti.

Consigliamo ai nuovi utenti che si stanno sviluppando localmente e che non dispongono di un metodo di autenticazione dal datore di lavoro di configurarlo. AWS IAM Identity Center Questo metodo include l'installazione di AWS CLI per facilitare la configurazione e per accedere regolarmente al portale di AWS accesso. Se scegli questo metodo, l'ambiente dovrebbe contenere i seguenti elementi dopo aver completato la procedura per l'autenticazione di IAM Identity Center nella Guida di riferimento agli AWS SDK e agli strumenti:

- La AWS CLI, che viene utilizzata per avviare una sessione del portale di accesso AWS prima di eseguire l'applicazione.

- Un AWS`config`file condiviso con un `[default]` profilo con un set di valori di configurazione a cui è possibile fare riferimento dall'SDK. Per trovare la posizione di questo file, consulta l'argomento relativo alla posizione dei file condivisi nella Guida di riferimento per SDK e strumenti AWS.
- Il `config` file condiviso imposta l'impostazione. `region` Questo imposta l'impostazione predefinita Regione AWS utilizzata dall'SDK per AWS le richieste. Questa regione viene utilizzata per le richieste di servizio SDK che non sono specificate con una regione da utilizzare.
- L'SDK utilizza la configurazione del provider di token SSO del profilo per acquisire le credenziali prima di inviare richieste a. AWS Il `sso_role_name` valore, che è un ruolo IAM connesso a un set di autorizzazioni IAM Identity Center, consente l'accesso ai dati Servizi AWS utilizzati nell'applicazione.

  Il seguente `config` file di esempio mostra un profilo predefinito impostato con la configurazione del provider di token SSO. L'`sso_session`impostazione del profilo si riferisce alla `sso-session`sezione denominata. La sezione `sso-session` contiene le impostazioni per avviare una sessione del portale di accesso AWS.

```
[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 v3 non necessita di pacchetti aggiuntivi (come `SSO` e`SSOOIDC`) da aggiungere all'applicazione per utilizzare l'autenticazione IAM Identity Center.

Per i dettagli sull'utilizzo esplicito di questo provider di credenziali, consulta il `fromSSO()`sito Web di npm (Node.js package manager).

Avvio di una sessione del portale di accesso AWS

Prima di eseguire un'applicazione che consente l'accessoServizi AWS, è necessaria una sessione attiva del portale di AWS accesso affinché l'SDK utilizzi l'autenticazione IAM Identity Center per

risolvere le credenziali. A seconda della durata della sessione configurata, l'accesso alla fine scadrà e l'SDK riscontrerà un errore di autenticazione. Per accedere al portale di accesso AWS, esegui il seguente comando nella AWS CLI.

```
aws sso login
```

Se hai seguito le istruzioni e disponi di una configurazione predefinita del profilo, non è necessario chiamare il comando con un'opzione. `--profile` Se la configurazione del provider di token SSO utilizza un profilo denominato, il comando è `aws sso login --profile named-profile`.

Per verificare facoltativamente se hai già una sessione attiva, esegui il AWS CLI comando seguente.

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

Se la sessione è attiva, la risposta a questo comando riporta l'account IAM Identity Center e il set di autorizzazioni configurati nel `config` file condiviso.

> **Note**
>
> Se hai già una sessione attiva del portale di accesso AWS e hai già eseguito `aws sso login`, non ti verrà richiesto di fornire credenziali.
> La procedura di accesso potrebbe richiederti di autorizzare l'accesso ai dati da parte della AWS CLI. Poiché AWS CLI è basato sull'SDK per Python, i messaggi di autorizzazione potrebbero contenere variazioni del `botocore` nome.

Ulteriori informazioni di autenticazione

Utenti umani, noti anche come identità umane, sono le persone, gli amministratori, gli sviluppatori, gli operatori e i consumatori delle tue applicazioni. Devono avere un'identità per accedere ai tuoi ambienti e alle tue applicazioni AWS. Gli utenti umani che sono membri della tua organizzazione, ovvero tu, lo sviluppatore, sono noti come identità della forza lavoro.

Utilizza credenziali temporanee per l'accesso. AWS È possibile utilizzare un provider di identità affinché gli utenti umani possano fornire l'accesso federato ad account AWS assumendo ruoli, che forniscono credenziali temporanee. Per la gestione centralizzata degli accessi, ti consigliamo di utilizzare AWS IAM Identity Center (IAM Identity Center) per gestire l'accesso ai tuoi account e le autorizzazioni all'interno di tali account. Per altre alternative, consulta quanto segue:

- Per ulteriori informazioni sulle best practice, consulta Best practice per la sicurezza in IAM nella Guida per l'utente di IAM.
- Per creare credenziali AWS a breve termine, consulta Credenziali di sicurezza temporanee in IAM nella Guida per l'utente di IAM.
- Per ulteriori informazioni su altri provider di credenziali AWS SDK for JavaScript V3, consulta Fornitori di credenziali standardizzati nella Guida di riferimento agli AWSSDK e agli strumenti.

Inizia con Node.js

Questa guida mostra come inizializzare un pacchetto NPM, aggiungere un client di servizio al pacchetto e utilizzare l' JavaScript SDK per avviare un'azione di servizio.

Lo scenario

Crea un nuovo pacchetto NPM con un file principale che esegua le seguenti operazioni:

- Crea un bucket Amazon Simple Storage Service
- Inserisce un oggetto nel bucket Amazon S3
- Legge l'oggetto nel bucket Amazon S3
- Conferma se l'utente desidera eliminare le risorse

Prerequisiti

Prima di eseguire l'esempio, è necessario effettuare le seguenti operazioni:

- Configura l'autenticazione SDK. Per ulteriori informazioni, consulta Autenticazione SDK con AWS.
- Installa Node.js.

Fase 1: Configurare la struttura dei pacchetti e installare i pacchetti client

Per configurare la struttura dei pacchetti e installare i pacchetti client:

1. Crea una nuova cartella `nodegetstarted` per contenere il pacchetto.
2. Dalla riga di comando, accedi alla nuova cartella.
3. Esegui il comando seguente per creare un `package.json` file predefinito:

```
npm init -y
```

4. Esegui il seguente comando per installare il pacchetto client Amazon S3:

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

5. Aggiungi `"type": "module"` al `package.json` file. Ciò indica a Node.js di utilizzare la moderna sintassi ESM. La versione finale `package.json` dovrebbe essere simile alla seguente:

```
{
  "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"
}
```

Passaggio 2: aggiungi le importazioni e il codice SDK necessari

Aggiungi il codice seguente a un file denominato `index.js` nella `nodegetstarted` cartella.

```
// 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();
}
```

Il codice di esempio può essere trovato qui GitHub.

Fase 3: Esegui l'esempio

> **Note**
>
> Ricordati di effettuare l'accesso! Se utilizzi IAM Identity Center per l'autenticazione, ricordati di accedere utilizzando il AWS CLI `aws sso login` comando.

1. Esegui `node index.js`.
2. Scegli se svuotare ed eliminare il bucket.
3. Se non elimini il bucket, assicurati di svuotarlo manualmente ed eliminarlo in un secondo momento.

Inizia nel browser

Questa sezione illustra un esempio che dimostra come eseguire la versione 3 (V3) dell'SDK for nel browser. JavaScript

> **Note**
>
> L'esecuzione di V3 nel browser è leggermente diversa dalla versione 2 (V2). Per ulteriori informazioni, consulta Utilizzo dei browser in V3.

Per altri esempi di utilizzo (V3) dell'SDK per, consulta. JavaScript SDK per esempi di JavaScript codice (v3)

Questo esempio di applicazione web mostra:

- Come accedere ai AWS servizi utilizzando Amazon Cognito per l'autenticazione.
- Come leggere un elenco di oggetti in un bucket Amazon Simple Storage Service (Amazon S3) utilizzando AWS Identity and Access Management un ruolo (IAM).

> **Note**
>
> Questo esempio non viene utilizzato AWS IAM Identity Center per l'autenticazione.

Lo scenario

Amazon S3 è un servizio di archiviazione di oggetti che offre scalabilità, disponibilità dei dati, sicurezza e prestazioni tra le migliori del settore. Puoi usare Amazon S3 per archiviare dati come oggetti all'interno di contenitori chiamati bucket. Per ulteriori informazioni su Amazon S3, consulta la Amazon S3 User Guide.

Questo esempio mostra come configurare ed eseguire un'app Web che presuppone un ruolo IAM per la lettura da un bucket Amazon S3. L'esempio utilizza la libreria front-end React e gli strumenti front-end Vite per fornire un ambiente di sviluppo. JavaScript L'app Web utilizza un pool di identità Amazon Cognito per fornire le credenziali necessarie per accedere ai servizi. AWS L'esempio di codice incluso illustra i modelli di base per il caricamento e l'utilizzo dell'SDK per JavaScript le app Web.

Fase 1: creare un pool di identità Amazon Cognito e un ruolo IAM

In questo esercizio, crei e utilizzi un pool di identità Amazon Cognito per fornire un accesso non autenticato alla tua app Web per il servizio Amazon S3. La creazione di un pool di identità crea anche un ruolo AWS Identity and Access Management (IAM) per supportare gli utenti guest non autenticati. In questo esempio, lavoreremo solo con il ruolo utente non autenticato per mantenere l'attività concentrata. È possibile integrare il supporto per un provider di identità e per gli utenti autenticati in un secondo momento. Per ulteriori informazioni sull'aggiunta di un pool di identità di Amazon Cognito, consulta il Tutorial: Creazione di un pool di identità nella Amazon Cognito Developer Guide.

Per creare un pool di identità Amazon Cognito e un ruolo IAM associato

1. Accedi AWS Management Console e apri la console Amazon Cognito all'indirizzo https://console.aws.amazon.com/cognito/.
2. Nel riquadro di navigazione a sinistra, scegli Identity pool.
3. Scegli Crea pool di identità.
4. In Configura la fiducia del pool di identità, scegli Accesso ospite per l'autenticazione degli utenti.
5. In Configura le autorizzazioni, scegli Crea un nuovo ruolo IAM e inserisci un nome (ad esempio, getStartedRole) nel nome del ruolo IAM.
6. In Configure properties, inserisci un nome (ad esempio, getStartedPool) in Identity pool name.
7. In Esamina e crea, conferma le selezioni effettuate per il nuovo pool di identità. Seleziona Modifica per tornare alla procedura guidata e modificare le eventuali impostazioni. Al termine, seleziona Crea un pool di identità.

8. Prendi nota dell'ID del pool di identità e della regione del pool di identità di Amazon Cognito appena creato. *Questi valori sono necessari per sostituire IDENTITY_POOL_ID e REGION in.* Passaggio 4: configura il codice del browser

Dopo aver creato il tuo pool di identità Amazon Cognito, sei pronto per aggiungere le autorizzazioni per Amazon S3 necessarie alla tua app web.

Passaggio 2: aggiungi una policy al ruolo IAM creato

Per abilitare l'accesso a un bucket Amazon S3 nella tua app Web, utilizza il ruolo IAM non autenticato (ad esempio, getStartedRole) creato per il tuo pool di identità Amazon Cognito (ad esempio,). getStartedPool Ciò richiede l'associazione di una policy IAM al ruolo. Per ulteriori informazioni sulla modifica dei ruoli IAM, consulta Modifica della politica di autorizzazione di un ruolo nella Guida per l'utente IAM.

Per aggiungere una policy Amazon S3 al ruolo IAM associato a utenti non autenticati

1. Accedi a AWS Management Console e apri la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nel pannello di navigazione a sinistra seleziona Ruoli.
3. Scegli il nome del ruolo che desideri modificare (ad esempio, getStartedRole), quindi scegli la scheda Autorizzazioni.
4. Scegli Aggiungi autorizzazioni, quindi scegli Allega politiche.
5. Nella pagina Aggiungi autorizzazioni per questo ruolo, trova e seleziona la casella di controllo per AmazonS3. ReadOnlyAccess

   > **Note**
   >
   > Puoi utilizzare questo processo per abilitare l'accesso a qualsiasi servizio. AWS

6. Scegli Aggiungi autorizzazioni.

Dopo aver creato il tuo pool di identità Amazon Cognito e aver aggiunto le autorizzazioni per Amazon S3 al tuo ruolo IAM per utenti non autenticati, sei pronto per aggiungere e configurare un bucket Amazon S3.

Fase 3: aggiungere un bucket e un oggetto Amazon S3

In questo passaggio, aggiungerai un bucket Amazon S3 e un oggetto per l'esempio. Attiverai anche la condivisione delle risorse tra le origini (CORS) per il bucket. Per ulteriori informazioni sulla creazione di bucket e oggetti Amazon S3, consulta la sezione Guida introduttiva ad Amazon S3 nella Amazon S3 User Guide.

Per aggiungere un bucket e un oggetto Amazon S3 con CORS

1. Accedere alla AWS Management Console e aprire la console Amazon S3 all'indirizzo https://console.aws.amazon.com/s3/.
2. Nel riquadro di navigazione a sinistra, scegli Bucket e scegli Crea bucket.
3. Inserisci un nome di bucket conforme alle regole di denominazione dei bucket (ad esempio, getstartedbucket) e scegli Crea bucket.
4. Scegli il bucket che hai creato, quindi scegli la scheda Oggetti. Scegliere quindi Upload (Carica).
5. In File e cartelle, seleziona Aggiungi file.
6. Seleziona un file da caricare, quindi scegli Apri. Quindi scegli Carica per completare il caricamento dell'oggetto nel tuo bucket.
7. Quindi, scegli la scheda Autorizzazioni del tuo bucket, quindi scegli Modifica nella sezione Cross-origin resource sharing (CORS). Inserisci il seguente codice JSON:

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

8. Seleziona Salvataggio delle modifiche.

Dopo aver aggiunto un bucket Amazon S3 e aggiunto un oggetto, sei pronto per configurare il codice del browser.

Passaggio 4: configura il codice del browser

L'applicazione di esempio è costituita da un'applicazione React a pagina singola. I file di questo esempio possono essere trovati qui su GitHub.

Per configurare l'applicazione di esempio

1. Installa Node.js.
2. Dalla riga di comando, clona il AWSCode Examples Repository:

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

3. Passa all'applicazione di esempio:

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

4. Eseguite il comando seguente per installare i pacchetti richiesti:

```
npm install
```

5. Quindi, apri `src/App.tsx` in un editor di testo e completa quanto segue:

   - Sostituisci *`YOUR_IDENTITY_POOL_ID con l'ID`* del pool di identità di Amazon Cognito in cui hai annotato. Fase 1: creare un pool di identità Amazon Cognito e un ruolo IAM
   - Sostituisci il valore per regione con la regione assegnata al tuo bucket Amazon S3 e al pool di identità Amazon Cognito. Tieni presente che le regioni per entrambi i servizi devono essere le stesse (ad esempio, us-east-2).
   - Sostituisci *`bucket-name con il nome del`* bucket in cui hai creato. Fase 3: aggiungere un bucket e un oggetto Amazon S3

Dopo aver sostituito il testo, salva il file. `App.tsx` Ora sei pronto per eseguire l'app web.

Passaggio 5: Esegui l'esempio

Per eseguire l'applicazione di esempio

1. Dalla riga di comando, accedete all'applicazione di esempio:

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

2. Dalla riga di comando, esegui il seguente comando:

```
npm run dev
```

   L'ambiente di sviluppo Vite verrà eseguito con il seguente messaggio:

```
VITE v4.3.9  ready in 280 ms

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

3. Nel tuo browser web, accedi all'URL mostrato sopra (ad esempio, http://localhost:5173). L'app di esempio ti mostrerà un elenco di nomi di file di oggetti nel tuo bucket Amazon S3.

Rimozione

Per ripulire le risorse create durante questo tutorial, procedi come segue:

- Nella console Amazon S3, elimina tutti gli oggetti e i bucket creati (ad esempio getstartedbucket).
- Nella console IAM, elimina il nome del ruolo (ad esempio,). getStartedRole
- Nella console Amazon Cognito, elimina il nome del pool di identità (ad esempio, getStartedPool).

Configura l'SDK per JavaScript

Gli argomenti di questa sezione spiegano come installare e caricare l'SDK JavaScript per accedere ai servizi Web supportati dall'SDK.

> **Note**
>
> Gli sviluppatori di React Native dovrebbero usare AWS Amplify per creare nuovi progetti su. AWS Vedi l'aws-sdk-react-nativearchivio per i dettagli.

Argomenti

Prerequisiti

Installa Node.js sui tuoi server, se non è già installato.

Argomenti

Configura un ambiente AWS Node.js

Per configurare un ambiente AWS Node.js in cui eseguire l'applicazione, utilizzate uno dei seguenti metodi:

- Scegli un'Amazon Machine Image (AMI) con Node.js preinstallato. Quindi crea un'istanza Amazon EC2 utilizzando quell'AMI. Quando crei la tua istanza Amazon EC2, scegli il tuo AMI tra. Marketplace AWS Marketplace AWS Cerca Node.js e scegli un'opzione AMI che includa una versione preinstallata di Node.js (32 o 64 bit).

- Crea un'istanza Amazon EC2 e installa Node.js su di essa. Per ulteriori informazioni su come installare Node.js su un'istanza Amazon Linux, consultaConfigurazione di Node.js su un'istanza Amazon EC2.
- Crea un ambiente serverless utilizzando AWS Lambda to run Node.js come funzione Lambda. Per ulteriori informazioni sull'utilizzo di Node.js all'interno di una funzione Lambda, consulta Programming model (Node.js) nella AWS Lambda Developer Guide.
- Distribuisci la tua applicazione Node.js su. AWS Elastic BeanstalkPer ulteriori informazioni sull'utilizzo di Node.js con Elastic Beanstalk, consulta Deploying Node.js AWS Elastic Beanstalk applications to nella Developer Guide.AWS Elastic Beanstalk
- Crea un server di applicazioni Node.js utilizzando. AWS OpsWorks Per ulteriori informazioni sull'utilizzo di Node.js con AWS OpsWorks, consulta Creazione del primo stack Node.js nella Guida per l'AWS OpsWorks utente.

Browser Web supportati

AWS SDK for JavaScript Supporta tutti i browser Web moderni.

Nella versione 3.183.0 o successiva, l'SDK JavaScript utilizza gli artefatti ES2020, che supportano le seguenti versioni minime.

Browser	Versione
Google Chrome	80,0 +
Mozilla Firefox	80,0+
Opera	63,0+
Microsoft Edge	80,0+
Apple Safari	14,1+
Samsung Internet	12,0+

Nella versione 3.182.0 o precedente, l'SDK JavaScript utilizza artefatti ES5, che supportano le seguenti versioni minime.

Browser	Versione
Google Chrome	49,0 o versioni successive
Mozilla Firefox	45,0+
Opera	36,0+
Microsoft Edge	12,0+
Windows Internet Explorer	N/D
Apple Safari	9,0+
Browser Android	76,0+
Browser UC	12.12+
Samsung Internet	5,0+

> **Note**
>
> Framework come questi AWS Amplify potrebbero non offrire lo stesso supporto per i browser dell'SDK. JavaScript Per i dettagli, consulta la AWS Amplify documentazione.

Installa l'SDK per JavaScript

Non tutti i servizi sono immediatamente disponibili nell'SDK o in tutte le AWS regioni.

Per installare un servizio AWS SDK for JavaScript utilizzando npm, il gestore di pacchetti Node.js, immettete il seguente comando al prompt dei comandi, dove *SERVICE* è il nome di un servizio, ad esempio. `s3`

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

Per un elenco completo dei pacchetti client del AWS SDK for JavaScript servizio, consulta la guida di riferimento AWS SDK for JavaScript API.

Carica l'SDK per JavaScript

Dopo aver installato l'SDK, puoi caricare un pacchetto client nell'applicazione del nodo utilizzando. `import` Ad esempio, per caricare il client Amazon S3 e il comando Amazon ListBucketsS3, usa quanto segue.

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

Configurare l'SDK per JavaScript

Prima di utilizzare l'SDK per JavaScript richiamare i servizi Web tramite l'API, è necessario configurare l'SDK. Come minimo, devi configurare:

- La AWS regione in cui richiederai i servizi
- In che modo il codice si autentica con AWS

Oltre a queste impostazioni, potresti dover configurare anche le autorizzazioni per le tue risorse. AWS Ad esempio, puoi limitare l'accesso a un bucket Amazon S3 o limitare una tabella Amazon DynamoDB per l'accesso in sola lettura.

La Guida di riferimento agli AWS SDK e agli strumenti contiene anche impostazioni, funzionalità e altri concetti fondamentali comuni a molti SDK. AWS

Gli argomenti di questa sezione descrivono i modi per configurare l'SDK JavaScript per Node.js e come JavaScript eseguirlo in un browser Web.

Argomenti

Configurazione per servizio

È possibile configurare l'SDK passando le informazioni di configurazione a un oggetto di servizio.

La configurazione a livello di servizio offre un controllo significativo sui singoli servizi, consentendo di aggiornare la configurazione dei singoli oggetti di servizio quando le esigenze variano rispetto alla configurazione predefinita.

> **Note**
>
> Nella versione 2.x della configurazione del AWS SDK for JavaScript servizio poteva essere passata ai singoli costruttori di client. Tuttavia, queste configurazioni verrebbero prima unite automaticamente in una copia della configurazione SDK globale. `AWS.config` Inoltre, richiamava `AWS.config.update({/* params */})` solo la configurazione aggiornata per i client di servizio istanziati dopo la chiamata di aggiornamento, non per i client esistenti.
>
> Questo comportamento era spesso fonte di confusione e rendeva difficile l'aggiunta di una configurazione all'oggetto globale che influiva solo su un sottoinsieme di client di servizio in modo compatibile con le versioni precedenti. Nella versione 3, non esiste più una configurazione globale gestita dall'SDK. La configurazione deve essere passata a ogni client di servizio istanziato. È ancora possibile condividere la stessa configurazione tra più client, ma tale configurazione non verrà unita automaticamente a uno stato globale.

Imposta la configurazione per servizio

Ogni servizio utilizzato nell'SDK per JavaScript è accessibile tramite un oggetto di servizio che fa parte dell'API per quel servizio. Ad esempio, per accedere al servizio Amazon S3, crei l'oggetto di servizio Amazon S3. È possibile specificare le impostazioni di configurazione specifiche per un servizio come parte del costruttore per quell'oggetto di servizio.

Ad esempio, se devi accedere agli oggetti Amazon EC2 in più AWS regioni, crea un oggetto di servizio Amazon EC2 per ogni regione e imposta di conseguenza la configurazione della regione di ciascun oggetto di servizio.

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

Imposta la regione AWS

Una AWS regione è un insieme denominato di AWS risorse nella stessa area geografica. Un esempio di regione è `us-east-1` la regione degli Stati Uniti orientali (Virginia settentrionale). Quando si crea un client di servizio nell'SDK, si specifica una regione in JavaScript modo che l'SDK acceda al servizio in quella regione. Alcuni servizi sono disponibili solo in regioni specifiche.

L'SDK per JavaScript non seleziona una regione per impostazione predefinita. Tuttavia, puoi impostare la AWS regione utilizzando una variabile di ambiente o un `config` file di configurazione condiviso.

In un costruttore di classi client

Quando si crea un'istanza di un oggetto servizio, è possibile specificare la AWS regione per quella risorsa come parte del costruttore della classe client, come illustrato di seguito.

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

Usa una variabile di ambiente

È possibile impostare la regione utilizzando la variabile di ambiente `AWS_REGION`. Se definisci questa variabile, l'SDK for la JavaScript legge e la usa.

Usa un file di configurazione condiviso

Proprio come il file delle credenziali condivise consente di archiviare le credenziali per l'utilizzo da parte dell'SDK, è possibile conservare la AWS regione e altre impostazioni di configurazione in un file condiviso denominato `config` SDK da utilizzare. Se la variabile di `AWS_SDK_LOAD_CONFIG` ambiente è impostata su un valore vero, l'SDK cerca JavaScript automaticamente un file al momento del caricamento. `config` Il percorso di salvataggio del file `config` varia a seconda del sistema operativo:

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

Se non si dispone già di un file condiviso `config`, è possibile crearne uno nella directory designata. In questo esempio il file `config` imposta sia la regione sia il formato di output.

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

Per ulteriori informazioni sull'utilizzo di `credentials` file `config` e file condivisi, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWS SDK e agli strumenti.

Ordine di precedenza per l'impostazione della regione

Di seguito è riportato l'ordine di precedenza per l'impostazione della regione:

1. Se una regione è passata a un costruttore della classe client, viene utilizzata quella regione.
2. Se una regione è impostata nella variabile di ambiente, viene utilizzata quella regione.
3. Altrimenti, viene utilizzata la regione definita nel file di configurazione condiviso.

Imposta le credenziali

AWS utilizza le credenziali per identificare chi sta chiamando i servizi e se è consentito l'accesso alle risorse richieste.

Sia che venga eseguito in un browser Web o in un server Node.js, il JavaScript codice deve ottenere credenziali valide prima di poter accedere ai servizi tramite l'API. Le credenziali possono essere impostate per servizio, passando le credenziali direttamente a un oggetto di servizio.

Esistono diversi modi per impostare le credenziali che differiscono tra Node.js e JavaScript nei browser Web. Gli argomenti in questa sezione descrivono come impostare le credenziali in Node.js o nei browser Web. In ogni caso, le opzioni sono riportate in ordine consigliato.

Procedure consigliate per le credenziali

L'impostazione corretta delle credenziali garantisce che l'applicazione o lo script di browser possano accedere ai servizi e alle risorse necessari, riducendo al minimo l'esposizione a problemi di sicurezza che possono inficiare le applicazioni mission critical o compromettere i dati sensibili.

Un importante principio da applicare durante l'impostazione delle credenziali è concedere sempre i privilegi minimi necessari per l'attività. È più sicuro fornire le autorizzazioni minime per le risorse e aggiungere ulteriori autorizzazioni in base alle esigenze, invece di fornire autorizzazioni che superano il privilegio minimo e, di conseguenza, dover risolvere i problemi di sicurezza scoperti più tardi. Ad esempio, a meno che non sia necessario leggere e scrivere singole risorse, come oggetti in un bucket Amazon S3 o una tabella DynamoDB, imposta tali autorizzazioni in modalità di sola lettura.

Per ulteriori informazioni sulla concessione del privilegio minimo, consulta la sezione Concedi il minimo privilegio dell'argomento Best Practices nella IAM User Guide.

Argomenti

Impostare le credenziali in Node.js

Consigliamo ai nuovi utenti che stanno sviluppando localmente e che non dispongono di un metodo di autenticazione dal datore di lavoro di AWS IAM Identity Center configurarsi. Per ulteriori informazioni, consulta Autenticazione SDK con AWS.

Vi sono diversi modi su Node.js per fornire le credenziali all'SDK. Alcuni di questi sono più sicuri e altri offrono più comodità durante lo sviluppo di un'applicazione. Quando ottenete le credenziali in Node.js, fate attenzione a non affidarvi a più di una fonte, ad esempio una variabile di ambiente e un file JSON che caricate. È possibile modificare le autorizzazioni sotto cui il codice viene eseguito senza realizzare che la modifica è avvenuta.

AWS SDK for JavaScript V3 fornisce una catena di provider di credenziali predefinita in Node.js, quindi non è necessario fornire un provider di credenziali in modo esplicito. La catena di provider di credenziali predefinita tenta di risolvere le credenziali da una varietà di fonti diverse con una determinata precedenza, finché non viene restituita una credenziale da una delle fonti. Puoi trovare la catena di fornitori di credenziali per SDK per V3 qui. JavaScript

Catena di fornitori di credenziali

Tutti gli SDK dispongono di una serie di posizioni (o fonti) che controllano per ottenere credenziali valide da utilizzare per effettuare una richiesta a un. Servizio AWS Dopo aver trovato credenziali valide, la ricerca viene interrotta. Questa ricerca sistematica è chiamata catena di fornitori di credenziali predefinita.

Per ogni fase della catena, esistono diversi modi per impostare i valori. L'impostazione dei valori direttamente nel codice ha sempre la precedenza, seguita dall'impostazione come variabili di ambiente e quindi nel AWS `config` file condiviso. Per ulteriori informazioni, consulta Precedenza delle impostazioni nella Guida di riferimento agli AWS SDK e agli strumenti.

La Guida di riferimento agli AWS SDK e agli strumenti contiene informazioni sulle impostazioni di configurazione SDK utilizzate da tutti gli AWS SDK e da. AWS CLI Per ulteriori informazioni su come configurare l'SDK tramite il AWS `config` file condiviso, consulta File di configurazione e credenziali condivisi. Per ulteriori informazioni su come configurare l'SDK tramite l'impostazione delle variabili di ambiente, consulta Supporto per le variabili di ambiente.

Con cui eseguire l'autenticazione AWS, AWS SDK for JavaScript controlla i fornitori di credenziali nell'ordine elencato nella tabella seguente.

AWS SDK for JavaScrip t Metodo del provider di credenziali API Reference in base alla precedenza	Fornitori di credenziali disponibili	AWS Guida di riferimento agli SDK e agli strumenti
`fromEnv()`	AWS chiavi di accesso dalle variabili di ambiente	AWS chiavi di accesso
`fromSSO()`	AWS IAM Identity Center. In questa guida, vediAutentica zione SDK con AWS.	Provider di credenziali IAM Identity Center
`fromIni()`	AWS chiavi di accesso da file condivisi `config` e `credentials` da file	AWS chiavi di accesso
	fornitore di entità affidabile (ad esempio`AWS_ROLE_ARN`)	Assumi un ruolo IAM
	Token di identità Web da AWS Security Token Service (AWS STS)	Federazione con identità web o OpenID Connect
	Credenziali Amazon Elastic Container Service (Amazon ECS)	Fornitore di credenziali per container
	Credenziali del profilo dell'ista nza Amazon Elastic Compute Cloud (Amazon EC2) (provider di credenziali IMDS)	Provider di credenziali IMDS
	Fornitore di credenziali di processo	Fornitore di credenziali di processo

AWS SDK for JavaScrip t Metodo del provider di credenziali API Reference in base alla precedenza	Fornitori di credenziali disponibili	AWS Guida di riferimento agli SDK e agli strumenti
	AWS IAM Identity Center credenziali	provider di credenziali IAM Identity Center
`fromProcess()`	Provider di credenziali di processo	Fornitore di credenziali di processo
`fromTokenFile()`	Token di identità Web da AWS Security Token Service ()AWS STS	Federazione con identità web o OpenID Connect
`fromContainerMetad ata()`	Credenziali Amazon Elastic Container Service (Amazon ECS)	Fornitore di credenziali per container
`fromInstanceMetada ta()`	Credenziali del profilo dell'ista nza Amazon Elastic Compute Cloud (Amazon EC2) (provider di credenziali IMDS)	Provider di credenziali IMDS

Se hai seguito l'approccio consigliato per i nuovi utenti per iniziare, configurerai AWS IAM Identity Center l'autenticazione durante l'argomento Autenticazione SDK con AWS Guida introduttiva. Altri metodi di autenticazione sono utili per diverse situazioni. Per evitare rischi per la sicurezza, consigliamo di utilizzare sempre credenziali a breve termine. Per altre procedure relative ai metodi di autenticazione, consulta Autenticazione e accesso nella Guida di riferimento agli AWS SDK e agli strumenti.

Gli argomenti in questa sezione descrivono come caricare le credenziali su Node.js.

Argomenti

- Carica le credenziali in Node.js dai ruoli IAM per Amazon EC2
- Caricare le credenziali per una funzione Lambda di Node.js

Carica le credenziali in Node.js dai ruoli IAM per Amazon EC2

Se esegui l'applicazione Node.js su un'istanza Amazon EC2, puoi sfruttare i ruoli IAM per Amazon EC2 per fornire automaticamente le credenziali all'istanza. Se configuri l'istanza per utilizzare i ruoli IAM, l'SDK seleziona automaticamente le credenziali IAM per l'applicazione, eliminando la necessità di fornire manualmente le credenziali.

Per ulteriori informazioni sull'aggiunta di ruoli IAM a un'istanza Amazon EC2, consulta Ruoli IAM per Amazon EC2.

Caricare le credenziali per una funzione Lambda di Node.js

Quando si crea una AWS Lambda funzione, è necessario creare un ruolo IAM speciale con il permesso di eseguire la funzione. Questo ruolo si chiama ruolo di esecuzione. Quando configuri una funzione Lambda, devi specificare il ruolo IAM che hai creato come ruolo di esecuzione corrispondente.

Il ruolo di esecuzione fornisce alla funzione Lambda le credenziali necessarie per eseguire e richiamare altri servizi Web. Di conseguenza, non è necessario fornire le credenziali per il codice Node.js scritto all'interno di una funzione Lambda.

Per ulteriori informazioni sulla creazione di un ruolo di esecuzione Lambda, consulta Manage permissions: Using an IAM role (execution role) nella Developer Guide.AWS Lambda

Impostare le credenziali in un browser Web

Vi sono diversi modi per fornire le credenziali all'SDK dagli script di browser. Alcuni di questi sono più sicuri e altri offrono più comodità durante lo sviluppo di uno script.

Ecco i modi in cui puoi fornire le tue credenziali, in ordine di raccomandazione:

1. Utilizzo di Amazon Cognito Identity per autenticare gli utenti e fornire credenziali
2. Utilizzo delle identità della federazione delle identità Web

> **Warning**
>
> Non è consigliabile codificare le AWS credenziali negli script. Effettuare l'hard coding delle credenziali pone un rischio di esposizione dell'ID chiave di accesso e della chiave di accesso segreta.

Argomenti

Usa Amazon Cognito Identity per autenticare gli utenti

Il modo consigliato per ottenere AWS le credenziali per gli script del browser consiste nell'utilizzare il client di credenziali di Amazon Cognito Identity. `CognitoIdentityClient` Amazon Cognito consente l'autenticazione degli utenti tramite provider di identità di terze parti.

Per utilizzare Amazon Cognito Identity, devi prima creare un pool di identità nella console Amazon Cognito. Un pool di identità rappresenta il gruppo di identità che l'applicazione fornisce agli utenti. Le identità fornite agli utenti identificano in modo univoco ogni account utente. Le identità di Amazon Cognito non sono credenziali. Vengono scambiate con credenziali utilizzando il supporto per la federazione delle identità web in (). AWS Security Token Service AWS STS

Amazon Cognito ti aiuta a gestire l'astrazione delle identità tra più provider di identità. L'identità caricata viene quindi scambiata con le credenziali in AWS STS.

Configurazione dell'oggetto credenziali di Amazon Cognito Identity

Se non ne hai ancora creato uno, crea un pool di identità da utilizzare con gli script del browser nella console Amazon Cognito prima di configurare il client Amazon Cognito. Crea e associa ruoli IAM autenticati e non autenticati per il tuo pool di identità. Per ulteriori informazioni, consulta Tutorial: Creazione di un pool di identità nella Amazon Cognito Developer Guide.

L'identità degli utenti non autenticati non viene verificata, il che rende questo ruolo appropriato per gli utenti ospiti della tua app o nei casi in cui non importa se la loro identità è stata verificata. Gli utenti autenticati accedono all'applicazione tramite un provider di identità di terza parte che verifica la loro identità. Assicurati di creare l'ambito delle autorizzazioni di risorse in modo appropriato per evitare che utenti non autenticati possano accedere a esse.

Dopo aver configurato un pool di identità, utilizza il `fromCognitoIdentityPool` metodo di `@aws-sdk/credential-providers` per recuperare le credenziali dal pool di identità. Nel seguente esempio di creazione di un client Amazon S3, sostituisci *`AWS_REGION con la regione e IDENTITY_POOL_ID con l'ID del pool di identità`*.

```
// 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 proprietà `logins` opzionale è una mappa di nomi di provider di identità ai token di identità per tali provider. Il modo in cui ottieni il token dal provider di identità dipende dal provider utilizzato. Ad esempio, se utilizzi un pool di utenti Amazon Cognito come provider di autenticazione, puoi utilizzare un metodo simile a quello riportato di seguito.

```
// 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;
};
```

Trasforma gli utenti non autenticati in utenti autenticati

Amazon Cognito supporta utenti autenticati e non autenticati. Gli utenti non autenticati ottengono l'accesso alle risorse anche se non sono connessi con un provider di identità. Tale livello di accesso è utile per visualizzare i contenuti agli utenti prima che effettuino l'accesso. Ogni utente non autenticato ha un'identità unica in Amazon Cognito anche se non è stato effettuato l'accesso e l'autenticazione individualmente.

Utente inizialmente non autenticato

Gli utenti in genere iniziano con il ruolo non autenticato, per cui è possibile impostare la proprietà delle credenziali dell'oggetto di configurazione senza una proprietà `logins`. In questo caso, le tue credenziali predefinite potrebbero essere le seguenti:

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

Cambio a utente autenticato

Quando un utente non autenticato accede a un provider di identità e disponi di un token, puoi passare da un utente non autenticato a uno autenticato chiamando una funzione personalizzata che aggiorna l'oggetto credenziali e aggiunge il 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;
```

```
}
```

Considerazioni su Node.js

Sebbene il codice Node.js lo sia JavaScript, l' AWS SDK for JavaScript utilizzo di Node.js può differire dall'utilizzo dell'SDK negli script del browser. Alcuni metodi API funzionano su Node.js ma non negli script di browser, e viceversa. E l'utilizzo corretto di alcune API dipende dal grado di familiarità con i modelli di codifica di Node.js comuni, ad esempio l'importazione e l'utilizzo di altri moduli di Node.js, come il modulo `File System (fs)`.

Utilizza i moduli Node.js integrati

Node.js fornisce una raccolta di moduli integrati che è possibile utilizzare senza installazione. Per utilizzare questi moduli, è necessario creare un oggetto con il metodo `require` per specificare il nome del modulo. Ad esempio, per includere il modulo HTTP integrato, utilizza il seguente metodo.

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

Richiama i metodi del modulo come se fossero metodi di quell'oggetto. Ad esempio, qui c'è il codice per leggere un file 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
  }
});
```

Per un elenco completo di tutti i moduli integrati forniti da Node.js, consultate la documentazione di Node.js sul sito Web Node.js.

Usa i pacchetti npm

Oltre ai moduli integrati, puoi anche includere e incorporare codice di terze parti proveniente dal `npm` gestore di pacchetti Node.js. Si tratta di un repository di pacchetti Node.js open source e di

un'interfaccia a riga di comando per installare i pacchetti. Per ulteriori informazioni npm e un elenco dei pacchetti attualmente disponibili, vedere https://www.npmjs.com. Puoi anche saperne di più sui pacchetti Node.js aggiuntivi che puoi usare qui GitHub.

Configurare MaxSockets in Node.js

Su Node.js, è possibile impostare il numero massimo di connessioni per origine. Se `maxSockets` è impostato, il client HTTP di basso livello mette in coda le richieste e le assegna ai socket non appena diventano disponibili.

In questo modo, è possibile impostare un limite superiore per il numero di richieste simultanee per una determinata origine effettuate alla volta. Impostando un valore basso è possibile ridurre il numero di errori di timeout o throttling ricevuti. Tuttavia, potrebbe aumentare l'utilizzo della memoria, perché le richieste vengono accodate fino a quando un socket diventa disponibile.

L'esempio seguente mostra come impostare `maxSockets` un client 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
  });
});
```

L'SDK for JavaScript utilizza `maxSockets` il valore 50 se non si fornisce un valore o un oggetto. `Agent` Se fornite un `Agent` oggetto, verrà utilizzato `maxSockets` il suo valore. Per ulteriori informazioni sull'impostazione `maxSockets` in Node.js, consultate la documentazione Node.js.

A partire dalla versione 3.521.0 di AWS SDK for JavaScript, è possibile utilizzare la seguente sintassi abbreviata per la configurazione. `requestHandler`

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

const client = new DynamoDBClient({
```

```
  requestHandler: {
    requestTimeout: 3_000,
    httpsAgent: { maxSockets: 25 },
  },
});
```

Riutilizza le connessioni con keep-alive in Node.js

L'agente HTTP/HTTPS Node.js predefinito crea una nuova connessione TCP per ogni nuova richiesta. Per evitare il costo di stabilire una nuova connessione, l'SDK for JavaScript riutilizza le connessioni TCP.

Per operazioni di breve durata, come le query Amazon DynamoDB, il sovraccarico di latenza dovuto alla configurazione di una connessione TCP potrebbe essere maggiore dell'operazione stessa. Inoltre, poiché la crittografia a riposo di DynamoDB è integrata AWS KMScon, è possibile che si verifichino delle latenze dovute al database che devono ristabilire AWS KMS nuove voci della cache per ogni operazione.

È inoltre possibile disattivare il mantenimento attivo di queste connessioni in base al singolo client di servizio, come illustrato nell'esempio seguente per un client DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "http";
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: new Agent({ keepAlive: false })
    })
});
```

Se `keepAlive` è abilitato, è anche possibile impostare il ritardo iniziale per i pacchetti TCP Keep-Alive con`keepAliveMsecs`, che per impostazione predefinita è 1000 ms. Vedere la documentazione di Node.js per i dettagli.

Configura i proxy per Node.js

Se non riesci a connetterti direttamente a Internet, l'SDK JavaScript supporta l'uso di proxy HTTP o HTTPS tramite un agente HTTP di terze parti.

Per trovare un agente HTTP di terze parti, cerca «proxy HTTP» su npm.

Per installare un agente proxy HTTP di terze parti, inserisci quanto segue al prompt dei comandi, dove *PROXY* è il nome del npm pacchetto.

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

Per utilizzare un proxy nell'applicazione, utilizzate la `httpsAgent` proprietà `httpAgent` and, come illustrato nell'esempio seguente per un client 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`non è uguale a`httpsAgent`, e poiché la maggior parte delle chiamate dal client sarà diretta a`https`, entrambe devono essere impostate.

Registra i pacchetti di certificati in Node.js

Gli archivi di fiducia predefiniti per Node.js includono i certificati necessari per accedere ai AWS servizi. In alcuni casi, può essere preferibile includere solo uno specifico set di certificati.

In questo esempio, un determinato certificato su disco viene utilizzato per creare un `https.Agent` che respinge le connessioni a meno che non venga fornito il certificato designato. Il nuovo file creato `https.Agent` viene quindi utilizzato dal client DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";
import { readFileSync } from "fs";
const certs = [readFileSync("/path/to/cert.pem")];
```

```
const agent = new Agent({
  rejectUnauthorized: true,
  ca: certs
});
const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  })
});
```

Considerazioni sullo script di browser

I seguenti argomenti descrivono considerazioni speciali sull'utilizzo degli script nei browser. AWS SDK for JavaScript

Argomenti

- Crea l'SDK per i browser
- Cross-Origin Resource Sharing (CORS)
- Raggruppa le applicazioni con webpack

Crea l'SDK per i browser

A differenza dell'SDK per la JavaScript versione 2 (V2), V3 non viene fornito come JavaScript file con supporto incluso per un set di servizi predefinito. V3 consente invece di raggruppare e includere nel browser solo l'SDK per i JavaScript file necessari, riducendo il sovraccarico. Ti consigliamo di utilizzare Webpack per raggruppare l'SDK richiesto per JavaScript i file e tutti i pacchetti di terze parti aggiuntivi richiesti in un unico `Javascript` file e caricarlo negli script del browser utilizzando un tag. `<script>` Per ulteriori informazioni su Webpack, consulta. Raggruppa le applicazioni con webpack Per un esempio che utilizza Webpack per caricare V3 SDK for JavaScript in un browser, vedi. Crea un'app per inviare dati a DynamoDB

Se lavori con l'SDK al di fuori di un ambiente che applica CORS nel tuo browser e desideri accedere a tutti i servizi forniti dall'SDK per JavaScript, puoi creare una copia personalizzata dell'SDK localmente clonando il repository ed eseguendo gli stessi strumenti di compilazione che creano la versione ospitata predefinita dell'SDK. Le sezioni seguenti descrivono la procedura per creare l'SDK con servizi e versioni dell'API aggiuntivi.

Usa SDK Builder per creare l'SDK per JavaScript

> **Note**
>
> La versione 3 (V3) di Amazon Web Services non supporta più Browser Builder. Per ridurre al minimo l'utilizzo della larghezza di banda delle applicazioni browser, ti consigliamo di importare moduli denominati e di raggrupparli per ridurne le dimensioni. Per ulteriori informazioni sul raggruppamento, consulta. Raggruppa le applicazioni con webpack

Cross-Origin Resource Sharing (CORS)

Il Cross-origin resource sharing, o CORS, è una caratteristica di sicurezza dei moderni browser Web. In questo modo i browser Web possono negoziare quali domini possono fare richieste di siti Web o servizi esterni.

CORS è fondamentale quando si sviluppano applicazioni di tipo browser con AWS SDK for JavaScript perché la maggior parte delle richieste di risorse vengono inviate a un dominio esterno, ad esempio l'endpoint di un servizio Web. Se JavaScript l'ambiente utilizza la sicurezza CORS, è necessario configurare CORS con il servizio.

CORS determina se consentire la condivisione di risorse in una richiesta tra origini diverse in base a quanto segue:

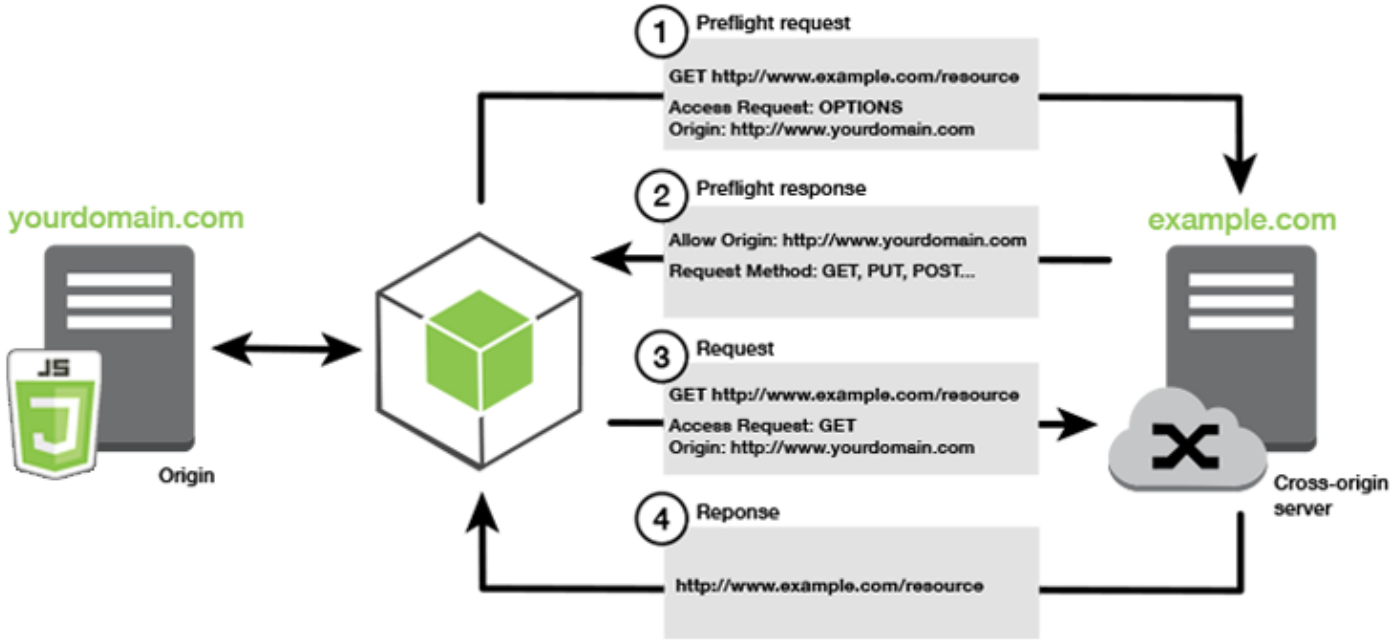
- Il dominio specifico che effettua la richiesta
- Il tipo di richiesta HTTP effettuata (GET, PUT, POST, DELETE e così via)

Come funziona CORS

Nel caso più semplice, lo script di browser invia una richiesta GET per una risorsa da un server in un altro dominio. A seconda della configurazione CORS del server, se la richiesta proviene da un dominio che è autorizzato a inviare le richieste GET, il server cross-origin risponde restituendo la risorsa richiesta.

Se il dominio che effettua la richiesta o il tipo di richiesta HTTP non è autorizzato, la richiesta viene negata. Tuttavia, CORS permette di preparare la richiesta prima dell'invio. In questo caso, viene effettuata una richiesta preliminare in cui viene inviata la richiesta di accesso `OPTIONS`. Se la configurazione CORS del server cross-origin consente di concedere l'accesso al dominio richiedente,

il server invia una risposta preliminare che elenca tutti i tipi di richieste HTTP che il dominio può effettuare sulla risorsa richiesta.



È richiesta la configurazione CORS?

I bucket Amazon S3 richiedono la configurazione CORS prima di poter eseguire operazioni su di essi. In alcuni JavaScript ambienti CORS potrebbe non essere applicato e pertanto la configurazione di CORS non è necessaria. Ad esempio, se ospiti l'applicazione da un bucket Amazon S3 e accedi alle risorse da `*.s3.amazonaws.com` o da qualche altro endpoint specifico, le tue richieste non accederanno a un dominio esterno. Pertanto, questa configurazione non richiede CORS. In questo caso, CORS viene ancora utilizzato per servizi diversi da Amazon S3.

Configurare CORS per un bucket Amazon S3

Puoi configurare un bucket Amazon S3 per utilizzare CORS nella console Amazon S3.

Se stai configurando CORS nella console di gestione dei servizi AWS Web, devi usare JSON per creare una configurazione CORS. La nuova console di gestione dei servizi AWS Web supporta solo le configurazioni JSON CORS.

> **Important**
>
> Nella nuova console di gestione dei servizi AWS Web, la configurazione CORS deve essere JSON.

1. Nella console di gestione dei servizi AWS Web, apri la console Amazon S3, trova il bucket che desideri configurare e seleziona la relativa casella di controllo.
2. Nel riquadro che si apre, scegli Autorizzazioni.
3. Nella scheda Autorizzazioni, scegliete Configurazione CORS.
4. Immettete la configurazione CORS nel CORS Configuration Editor, quindi scegliete Salva.

Una configurazione CORS è un file XML che contiene una serie di regole all'interno di un `<CORSRule>`. Una configurazione può avere massimo 100 regole. Una regola è definita da uno dei seguenti tag:

- `<AllowedOrigin>`— Specificate le origini del dominio a cui consentite di effettuare richieste tra domini.
- `<AllowedMethod>`— Specificate il tipo di richiesta consentita (GET, PUT, POST, DELETE, HEAD) nelle richieste tra domini.
- `<AllowedHeader>`— Speciifica le intestazioni consentite in una richiesta di preflight.

Per esempi di configurazioni, vedi Come posso configurare CORS sul mio bucket? nella Guida per l'utente di Amazon Simple Storage Service.

Esempio di configurazione CORS

Il seguente esempio di configurazione CORS consente a un utente di visualizzare, aggiungere, rimuovere o aggiornare oggetti all'interno di un bucket dal dominio. `example.org` Tuttavia, ti consigliamo di estendere l'ambito `<AllowedOrigin>` al dominio del tuo sito web. È possibile specificare `"*"` per consentire l'origine.

> **Important**
>
> Nella nuova console S3, la configurazione CORS deve essere JSON.

XML

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

JSON

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

Questa configurazione non autorizza l'utente a eseguire azioni nel bucket. Abilita il modello di sicurezza del browser per consentire una richiesta ad Amazon S3. Le autorizzazioni devono essere configurate tramite i permessi dei bucket o i permessi dei ruoli IAM.

Puoi utilizzarlo `ExposeHeader` per consentire all'SDK di leggere le intestazioni di risposta restituite da Amazon S3. Ad esempio, se leggi l'`ETag`intestazione da un caricamento in più parti `PUT` o in più parti, devi includere il `ExposeHeader` tag nella configurazione, come mostrato nell'esempio precedente. Il kit SDK è in grado di accedere solo alle intestazioni esposte attraverso la configurazione CORS. Se si impostano i metadati nell'oggetto, i valori vengono restituiti come intestazioni con il prefisso `x-amz-meta-`, ad esempio `x-amz-meta-my-custom-header`, e devono essere esposti in modo analogo.

Raggruppa le applicazioni con webpack

L'uso di moduli di codice da parte delle applicazioni Web negli script del browser o in Node.js crea dipendenze. Questi moduli di codice possono avere dipendenze proprie, generando una raccolta di moduli interconnessi richiesti dall'applicazione per funzionare. Per gestire le dipendenze, puoi usare un bundler di moduli come. `webpack`

Il bundler di `webpack` moduli analizza il codice dell'applicazione, cercando le nostre `require` istruzioni, `import` per creare pacchetti che contengono tutte le risorse di cui l'applicazione ha bisogno. In questo modo le risorse possono essere facilmente servite tramite una pagina Web. L'SDK for JavaScript può essere incluso `webpack` come una delle dipendenze da includere nel pacchetto di output.

Per ulteriori informazioni in meritowebpack, consulta il bundler del modulo webpack su. GitHub

Installa webpack

Per installare il bundler del `webpack` modulo, devi prima avere installato npm, il gestore di pacchetti Node.js. Digita il comando seguente per installare la `webpack` CLI e JavaScript il modulo.

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

Per utilizzare il `path` modulo per lavorare con i percorsi di file e directory, che viene installato automaticamente con webpack, potrebbe essere necessario installare il pacchetto Node.js`path-browserify`.

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

Configura webpack

Per impostazione predefinita, Webpack cerca un JavaScript file denominato `webpack.config.js` nella directory principale del progetto. Questo file specifica le opzioni di configurazione. Di seguito è riportato un esempio di file di `webpack.config.js` configurazione per la WebPack versione 5.0.0 e successive.

> **Note**
>
> I requisiti di configurazione di Webpack variano a seconda della versione di Webpack installata. Per ulteriori informazioni, consulta la documentazione di 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"}]
  }
  **/
};
```

In questo esempio, `browser.js` viene specificato come punto di ingresso. Il punto di ingresso è il file `webpack` utilizzato per iniziare la ricerca dei moduli importati. Come `bundle.js` è specificato il

nome del file di output. Questo file di output conterrà tutto JavaScript il necessario per l'esecuzione dell'applicazione. Se il codice specificato nel punto di ingresso importa o richiede altri moduli, come l'SDK for JavaScript, quel codice viene fornito in bundle senza bisogno di specificarlo nella configurazione.

Esegui webpack

Per creare un'applicazione da usare`webpack`, aggiungi quanto segue all'`scripts`oggetto nel tuo `package.json` file.

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

Di seguito è riportato un `package.json` file di esempio che dimostra l'aggiunta`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"
  }
}
```

Per creare la tua applicazione, inserisci il seguente comando.

```
npm run build
```

Il bundler del `webpack` modulo genera quindi il JavaScript file specificato nella directory principale del progetto.

Usa il pacchetto webpack

Per utilizzare il pacchetto in uno script del browser, puoi incorporarlo utilizzando un `<script>` tag, come mostrato nell'esempio seguente.

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

Bundle per Node.js

È possibile `webpack` utilizzarlo per generare pacchetti che vengono eseguiti in Node.js specificandolo `node` come destinazione nella configurazione.

```
target: "node"
```

Questa funzione è utile quando si esegue un'applicazione Node.js in un ambiente in cui lo spazio su disco è limitato. Ecco un esempio di configurazione `webpack.config.js` con Node.js specificato come destinazione dell'output.

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

Lavora con AWS i servizi nell'SDK per JavaScript

La AWS SDK for JavaScript versione v3 fornisce l'accesso ai servizi che supporta tramite una raccolta di classi client. Da queste classi client, si creano oggetti di interfaccia di servizio, comunemente chiamati oggetti di servizio. Ogni AWS servizio supportato ha una o più classi client che offrono API di basso livello per l'utilizzo delle funzionalità e delle risorse del servizio. Ad esempio, le API di Amazon DynamoDB sono disponibili tramite la classe. `DynamoDB`

I servizi esposti tramite l'SDK JavaScript seguono lo schema di richiesta-risposta per lo scambio di messaggi con le applicazioni chiamanti. In questo modello, il codice che richiama un servizio inoltra una richiesta HTTP/HTTPS a un endpoint per il servizio. La richiesta contiene i parametri necessari per richiamare correttamente la funzionalità specifica chiamata. Il servizio richiamato genera una risposta che viene inviata nuovamente al richiedente. La risposta contiene dati se l'operazione ha avuto esito positivo o informazioni di errore se l'operazione non ha avuto esito positivo.

L'invocazione AWS di un servizio include l'intero ciclo di vita di richiesta e risposta di un'operazione su un oggetto di servizio, inclusi eventuali tentativi di nuovo tentativo. Una richiesta contiene zero o più proprietà come parametri JSON. La risposta è incapsulata in un oggetto correlato all'operazione e viene restituita al richiedente tramite una delle diverse tecniche, ad esempio una funzione di callback o una promessa. JavaScript

Argomenti

Crea e richiama oggetti di servizio

L' JavaScript API supporta la maggior parte dei AWS servizi disponibili. Ogni servizio dell' JavaScriptAPI fornisce a una classe client un `send` metodo da utilizzare per richiamare tutte le API supportate dal servizio. Per ulteriori informazioni sulle classi di servizio, le operazioni e i parametri nell' JavaScript API, consulta l'API Reference.

Quando si utilizza l'SDK in Node.js, si aggiunge il pacchetto SDK per ogni servizio necessario all'utilizzo dell'applicazione`import`, che fornisce supporto per tutti i servizi correnti. L'esempio seguente crea un oggetto di servizio Amazon S3 nella `us-west-1` regione.

```
// 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"
  });
```

Specificare i parametri dell'oggetto di servizio

Quando chiami un metodo di un oggetto di servizio, trasferisci i parametri in JSON come richiesto dall'API. Ad esempio, in Amazon S3, per ottenere un oggetto per un bucket e una chiave specificati, passa i seguenti parametri al `GetObjectCommand` metodo da. `S3Client` Per ulteriori informazioni sul trasferimento di parametri JSON, consulta Lavora con JSON.

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

Per ulteriori informazioni sui parametri di Amazon S3, consulta @aws -sdk/client-s3 nell'API Reference.

Chiama i servizi in modo asincrono

Tutte le richieste effettuate attraverso l'SDK sono asincrone. Questo è importante da tenere a mente quando si scrivono gli script del browser. JavaScript l'esecuzione in un browser Web in genere ha un solo thread di esecuzione. Dopo aver effettuato una chiamata asincrona a un AWS servizio, lo script del browser continua a essere eseguito e durante il processo può provare a eseguire il codice che dipende da quel risultato asincrono prima che venga restituito.

L'esecuzione di chiamate asincrone a un AWS servizio include la gestione di tali chiamate in modo che il codice non tenti di utilizzare i dati prima che i dati siano disponibili. Gli argomenti di questa sezione spiegano la necessità di gestire le chiamate asincrone e illustrano diverse tecniche che è possibile utilizzare per gestirle.

Sebbene sia possibile utilizzare una qualsiasi di queste tecniche per gestire le chiamate asincrone, si consiglia di utilizzare async/await per tutto il nuovo codice.

async/await

Si consiglia di utilizzare questa tecnica in quanto è il comportamento predefinito in V3.

promettere

Usa questa tecnica nei browser che non supportano async/await.

richiamata

Evita di usare i callback tranne in casi molto semplici. Tuttavia, potresti trovarlo utile per gli scenari di migrazione.

Argomenti

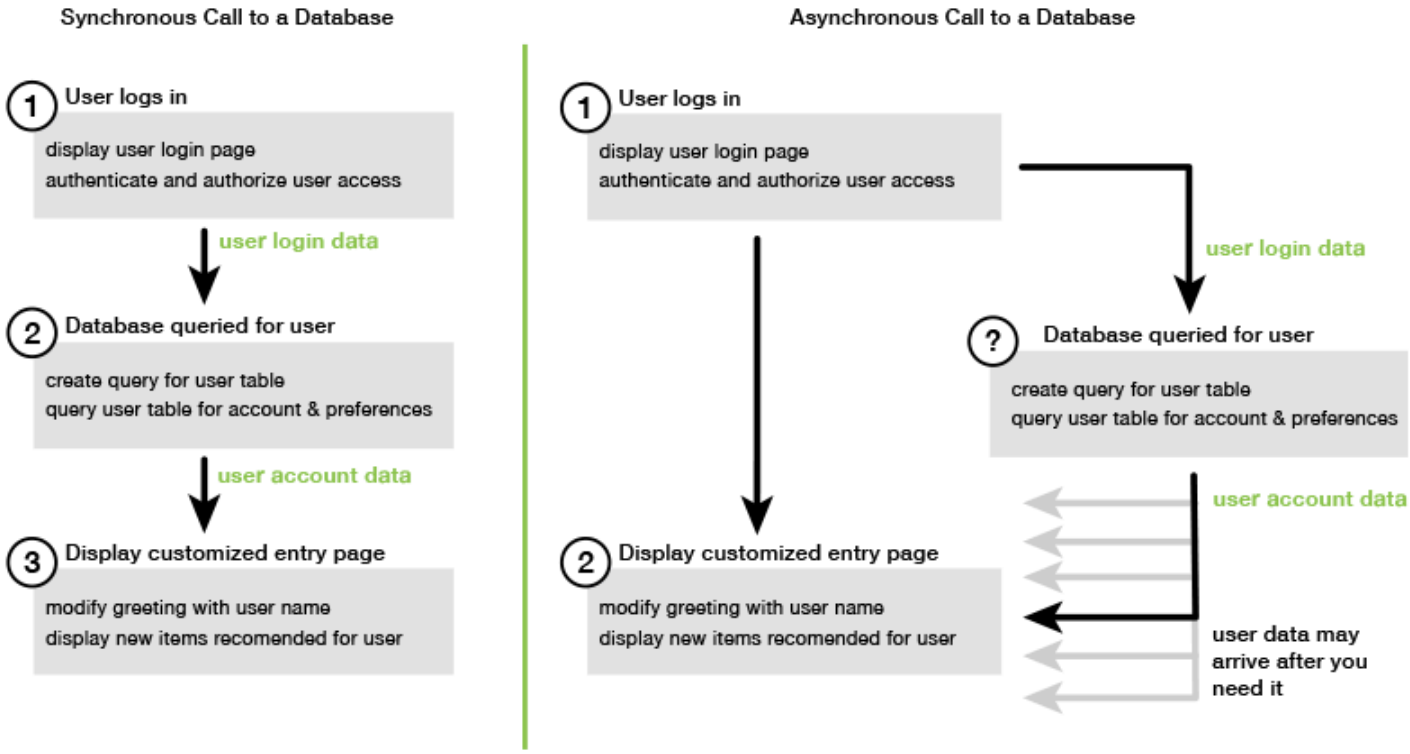
Gestisci le chiamate asincrone

Ad esempio, la home page di un sito Web di e-commerce consente l'accesso ai clienti registrati. Parte del vantaggio per i clienti che effettuano l'accesso è che, dopo l'accesso, il sito si adatta alle loro preferenze specifiche. Per fare in modo che ciò accada:

1. Il cliente deve accedere ed essere convalidato con le proprie credenziali di accesso.
2. Le preferenze del cliente vengono richieste da un database dei clienti.
3. Il database fornisce le preferenze del cliente che vengono utilizzate per personalizzare il sito prima che la pagina venga caricata.

Se queste attività vengono eseguite in modo sincrono, ognuna deve terminare prima venga avviata quella successiva. Il caricamento della pagina web non potrà essere completato finché le preferenze del cliente non torneranno dal database. Tuttavia, dopo che la query del database viene inviata al server, la ricezione dei dati del cliente può essere posticipata o può addirittura fallire a causa di colli di bottiglia della rete, traffico di database eccezionalmente elevato o connessione scadente dei dispositivi mobili.

Per evitare che il sito Web si blocchi in tali condizioni, chiama il database in modo asincrono. Dopo l'esecuzione della chiamata al database, inviando la tua richiesta asincrona, il tuo codice continua a essere eseguito come previsto. Se non gestisci correttamente la risposta di una chiamata asincrona, il tuo codice può tentare di utilizzare le informazioni che si aspetta dal database quando tali dati non sono ancora disponibili.



Usa async/await

Piuttosto che usare le promesse, dovresti considerare l'uso di async/await. Le funzioni asincrone sono più semplici e richiedono meno boilerplate rispetto all'uso delle promesse. Await può essere utilizzato solo in una funzione asincrona per attendere in modo asincrono un valore.

L'esempio seguente utilizza async/await per elencare tutte le tabelle Amazon DynamoDB in. `us-west-2`

> **Note**
>
> Per eseguire questo esempio:
>
> - Installa il client AWS SDK for JavaScript DynamoDB `npm install @aws-sdk/client-dynamodb` accedendo alla riga di comando del tuo progetto.

- Assicurati di aver configurato correttamente le tue AWS credenziali. Per ulteriori informazioni, consulta Imposta le credenziali.

```
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**
>
> Non tutti i browser supportano async/await. Vedi Funzioni asincrone per un elenco di browser con supporto async/await.

JavaScript Usa le promesse

Utilizza il metodo AWS SDK for JavaScript v3 (`ListTablesCommand`) del client di servizio per effettuare la chiamata di servizio e gestire il flusso asincrono invece di utilizzare i callback. L'esempio seguente mostra come inserire i nomi delle tabelle Amazon DynamoDB. `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 più promesse

In alcune situazioni, il tuo codice deve effettuare più chiamate asincrone che richiedono un intervento solo quando sono state restituite tutte correttamente. Se gestisci tali chiamate singole al metodo asincrono con le promesse, puoi creare una promessa aggiuntiva che utilizza il metodo `all`.

Questo metodo soddisfa questa promessa universale se e quando le promesse che trasferisci al metodo vengono soddisfatte. Alla funzione di callback viene trasferito una matrice dei valori delle promesse trasferite al metodo `all`.

Nell'esempio seguente, una AWS Lambda funzione deve effettuare tre chiamate asincrone ad Amazon DynamoDB, ma può essere completata solo dopo aver soddisfatto le promesse per ogni chiamata.

```
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;
```

Browser e Node.js supportano le promesse

Il supporto per le JavaScript promesse native (ECMAScript 2015) dipende dal JavaScript motore e dalla versione in cui viene eseguito il codice. Per determinare il supporto per le JavaScript promesse in ogni ambiente in cui il codice deve essere eseguito, consulta la tabella di compatibilità ECMAScript su. GitHub

Usa una funzione di callback anonima

Ogni metodo dell'oggetto di servizio può accettare una funzione di callback anonima come ultimo parametro. La firma di questa funzione di callback è la seguente.

```
function(error, data) {
```

```
    // callback handling code
};
```

Questa funzione di callback viene eseguita quando vengono restituiti una risposta corretta o i dati dell'errore. Se la chiamata al metodo va a buon fine, i contenuti della risposta sono disponibili per la funzione di callback nel parametro `data`. Se la chiamata non va a buon fine, i dettagli sull'errore vengono forniti nel parametro `error`.

In genere il codice all'interno della funzione di callback verifica un errore, che elabora nel caso venga restituito. Se non viene restituito un errore, il codice recupera i dati dalla risposta dal parametro `data`. Il formato di base della funzione di callback è simile al seguente esempio.

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

Nell'esempio precedente, i dettagli dell'errore o dei dati restituiti vengono registrati nella console. Ecco un esempio che mostra una funzione di callback trasferita come parte della chiamata a un metodo su un oggetto di servizio.

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

Crea richieste ai clienti di servizio

Effettuare richieste ai clienti del AWS servizio è semplice. La versione 3 (V3) dell'SDK JavaScript consente di inviare richieste.

> **Note**
>
> È inoltre possibile eseguire operazioni utilizzando i comandi della versione 2 (V2) quando si utilizza la versione 3 dell'SDK for. JavaScript Per ulteriori informazioni, consulta Utilizzo dei comandi V2.

Per inviare una richiesta:

1. Inizializza un oggetto client con la configurazione desiderata, ad esempio una AWS regione specifica.
2. (Facoltativo) Crea un oggetto JSON di richiesta con i valori per la richiesta, ad esempio il nome di uno specifico bucket Amazon S3. Puoi esaminare i parametri della richiesta consultando l'argomento API Reference relativo all'interfaccia con il nome associato al metodo client. Ad esempio, se si utilizza il metodo *`AbcCommand`*client, l'interfaccia di richiesta è *`AbcInput`*.
3. Inizializza un comando di servizio, facoltativamente, con l'oggetto di richiesta come input.
4. Chiama `send` il client con l'oggetto comando come input.

Ad esempio, per elencare le tue tabelle Amazon DynamoDB, puoi farlo con `us-west-2` 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);
  }
})();
```

Gestisci le risposte dei clienti di assistenza

Dopo che un metodo client di servizio è stato chiamato, restituisce un'istanza dell'oggetto di risposta di un'interfaccia con il nome associato al metodo client. Ad esempio, se si utilizza il metodo *`AbcCommand`*client, l'oggetto di risposta è di tipo *`AbcResponse`*(interfaccia).

Accedi ai dati restituiti nella risposta

L'oggetto di risposta contiene i dati, come proprietà, restituiti dalla richiesta di servizio.

NelCrea richieste ai clienti di servizio, il `ListTablesCommand` comando ha restituito i nomi delle tabelle nella `TableNames` proprietà della risposta.

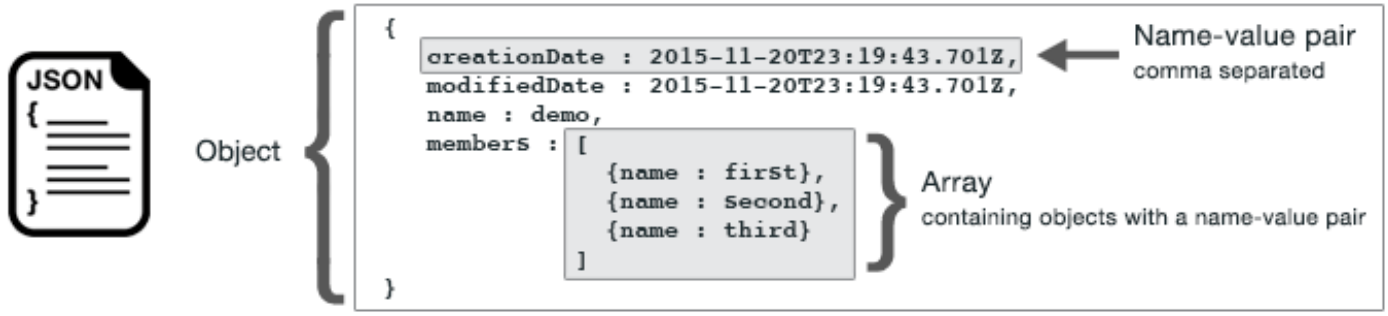
Informazioni sugli errori di accesso

Se un comando fallisce, genera un'eccezione. È possibile gestire l'eccezione in base alle proprie esigenze.

Lavora con JSON

JSON è un formato per lo scambio di dati leggibile sia dall'uomo che dalla macchina. Sebbene il nome JSON sia l'acronimo di JavaScript Object Notation, il formato di JSON è indipendente da qualsiasi linguaggio di programmazione.

AWS SDK for JavaScript Utilizza JSON per inviare dati agli oggetti di servizio quando effettua richieste e riceve dati dagli oggetti di servizio come JSON. Per ulteriori informazioni su JSON, consulta json.org.



JSON rappresenta i dati in due modi:

- Come oggetto, che è una raccolta non ordinata di coppie nome-valore. Un oggetto viene definito all'interno di parentesi graffe sinistra (`{`) e destra (`}`). Ogni coppia nome-valore inizia con il nome, seguita dai due punti e dal valore. Le coppie nome-valore sono separate da virgole.
- Come matrice, che è una raccolta ordinata di valori. Una matrice viene definita all'interno di parentesi quadre sinistra (`[`) e destra (`]`). Gli elementi nella matrice sono separati da virgole.

Ecco un esempio di un oggetto JSON che contiene una matrice di oggetti in cui gli oggetti rappresentano le carte in un gioco di carte. Ogni carta è definita da due coppie nome-valore, una che specifica un valore univoco per identificare quella carta e un'altra che specifica un URL che punta all'immagine della carta corrispondente.

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

JSON come parametri dell'oggetto di servizio

Ecco un esempio di JSON semplice utilizzato per definire i parametri di una chiamata a un oggetto di AWS Lambda servizio.

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

L'oggetto `params` è definito da tre coppie nome-valore, separate da virgole e racchiuse fra parentesi graffe sinistra e destra. Quando si forniscono i parametri a una chiamata al metodo dell'oggetto di servizio, i nomi vengono determinati dai nomi dei parametri per il metodo dell'oggetto di servizio che si intende chiamare. Quando si richiama una funzione Lambda`FunctionName`,`Payload`, `LogType` e sono i parametri utilizzati per chiamare il metodo su un oggetto `invoke` del servizio Lambda.

Quando passate parametri a una chiamata al metodo dell'oggetto servizio, fornite l'oggetto JSON alla chiamata al metodo, come illustrato nel seguente esempio di richiamo di una funzione 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 per esempi di JavaScript codice

Gli argomenti di questa sezione contengono esempi di come utilizzare AWS SDK for JavaScript le API di vari servizi per eseguire attività comuni.

Trovate il codice sorgente per questi e altri esempi nel AWS Code Examples Repository su. GitHub Per proporre un nuovo esempio di codice che il team addetto alla AWS documentazione possa prendere in considerazione la produzione, crea una richiesta. Il team sta cercando di produrre esempi di codice che coprano scenari e casi d'uso più ampi rispetto ai semplici frammenti di codice che coprono solo le singole chiamate API. Per istruzioni, consultate la sezione Codice di creazione nelle linee guida per i contributi su GitHub.

Important

Questi esempi utilizzano la sintassi di importazione/esportazione ECMAScript6.

- Ciò richiede la versione 14.17 o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
- Se preferisci utilizzare la sintassi CommonJS, consulta le linee guida JavaScript Sintassi ES6/CommonJS per la conversione.

Argomenti

- JavaScript Sintassi ES6/CommonJS

- Esempi di Amazon DynamoDB
- Esempi di AWS Elemental MediaConvert
- Esempi di AWS Lambda
- Esempi di Amazon Lex
- Esempi di Amazon Polly
- Esempi di Amazon Redshift
- Esempi di Amazon Simple Email Service
- Esempi di servizi di notifica Amazon Simple
- Esempi di Amazon Transcribe
- Configurazione di Node.js su un'istanza Amazon EC2
- Crea un'app per inviare dati a DynamoDB
- Crea un'app di trascrizione con utenti autenticati
- Richiamare Lambda con API Gateway
- Creazione di AWS flussi di lavoro serverless utilizzando AWS SDK for JavaScript
- Creazione di eventi pianificati per eseguire AWS Lambda funzioni
- Creazione di un chatbot Amazon Lex
- Creazione di un'applicazione di messaggistica di esempio

JavaScript Sintassi ES6/CommonJS

Gli esempi diAWS SDK for JavaScript codice sono scritti in ECMAScript 6 (ES6). ES6 offre una nuova sintassi e nuove funzionalità per rendere il codice più moderno e leggibile e fare di più.

ES6 richiede l'utilizzo di Node.js versione 13.x o successiva. Per scaricare e installare e installare e installare e installare e installare la versione più recente di Node.js, consultare le fasi riportate di Node.js. Tuttavia, puoi convertire uno qualsiasi dei nostri esempi nella sintassi CommonJS utilizzando le fasi riportate di seguito:

- Rimuovilo`"type" : "module"`package.json dall'ambiente del tuo progetto.
- Converti tutte le istruzioni ES6 in`import` istruzioni CommonJS`require`. Ad esempio, converti:

```
import { CreateBucketCommand } from "@aws-sdk/client-s3";
```

```
import { s3 } from "./libs/s3Client.js";
```

Al suo equivalente CommonJS:

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

- Converti tutte le istruzioni ES6 in`export` istruzioni CommonJS`module.exports`. Ad esempio, converti:

```
export {s3}
```

Al suo equivalente CommonJS:

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

L'esempio seguente mostra l'esempio di codice per la creazione di un bucket Amazon S3 sia in ES6 che in 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();
```

Esempi di Amazon DynamoDB

Amazon DynamoDB è un database cloud NoSQL completamente gestito che supporta modelli di archiviazione di documenti e chiave-valore. È possibile creare tabelle prive di schema per i dati senza la necessità di allestire o gestire i server di database dedicati.

L' JavaScript API per DynamoDB è esposta tramite `DynamoDB` le classi`DynamoDBStreams`, `DynamoDB.DocumentClient` e client. Per ulteriori informazioni sull'utilizzo delle classi client DynamoDB, vedere Class: DynamoDB, Class: DynamoDBStreams e Class: DynamoDB Utility nell'API Reference.

Argomenti

- Creazione e utilizzo di tabelle in DynamoDB

- Lettura e scrittura di un singolo elemento in DynamoDB
- Lettura e scrittura di elementi in batch in DynamoDB
- Interrogazione e scansione di una tabella DynamoDB
- Utilizzo del DynamoDB Document Client

Creazione e utilizzo di tabelle in DynamoDB

Questo esempio di codice di Node.js illustra:

- Come creare e gestire le tabelle utilizzate per archiviare e recuperare dati da DynamoDB.

Lo scenario

Analogamente ad altri sistemi di database, DynamoDB archivia i dati in tabelle. Una tabella DynamoDB è una raccolta di dati organizzata in elementi analoghi alle righe. Per archiviare o accedere ai dati in DynamoDB, devi creare e lavorare con le tabelle.

In questo esempio, si utilizza una serie di moduli Node.js per eseguire operazioni di base con una tabella DynamoDB. Il codice utilizza l'SDK per JavaScript creare e lavorare con le tabelle utilizzando questi metodi della `DynamoDB` classe client:

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

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi esempi di Node.js e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.

- Installa SDK per il client JavaScript DynamoDB. Per ulteriori informazioni, consulta Cosa c'è di nuovo nella versione 3.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli SDK e agli AWS strumenti.

Important

Questi esempi utilizzano ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Note

Per informazioni sui tipi di dati utilizzati in questi esempi, consulta Tipi di dati e regole di denominazione supportati in Amazon DynamoDB.

Creazione di una tabella

Crea un modulo Node.js con il nome del file `create-table.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Crea un oggetto JSON che contiene i parametri necessari per creare una tabella, che in questo esempio include il nome e il tipo di dati per ogni attributo, lo schema chiave, il nome della tabella e le unità di throughput sui cui effettuare il provisioning. Chiama il `CreateTableCommand` metodo dell'oggetto servizio 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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Elencare i tavoli

Crea un modulo Node.js con il nome del file `list-tables.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Creare un oggetto JSON che contiene i parametri necessari per elencare le tabelle, che in questo esempio limita il numero di tabelle elencate a 10. Chiama il `ListTablesCommand` metodo dell'oggetto servizio 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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Descrizione di una tabella

Crea un modulo Node.js con il nome del file `describe-table.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea DynamoDB un oggetto servizio client. Crea un oggetto JSON contenente i parametri necessari per descrivere un `DescribeTableCommand` metodo dell'oggetto servizio 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;
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione di una tabella

Crea un modulo Node.js con il nome del file `delete-table.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Crea un oggetto JSON che contiene i parametri necessari per eliminare una tabella, che in questo esempio include il nome della tabella fornito come parametro della riga di comando. Chiama il `DeleteTableCommand` metodo dell'oggetto servizio 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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Lettura e scrittura di un singolo elemento in DynamoDB

Questo esempio di codice di Node.js illustra:

- Come aggiungere un elemento in una tabella DynamoDB.

- Come recuperare un elemento in una tabella DynamoDB.
- Come eliminare un elemento in una tabella DynamoDB.

Lo scenario

In questo esempio, si utilizza una serie di moduli Node.js per leggere e scrivere un elemento in una tabella DynamoDB utilizzando questi metodi della `DynamoDB` classe client:

- PutItemCommand
- UpdateItemCommand
- GetItemCommand
- DeleteItemCommand

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi esempi di Node.js e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB, vedere. Creazione e utilizzo di tabelle in DynamoDB

> **Important**
>
> Questi esempi utilizzano ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

> **Note**
>
> Per informazioni sui tipi di dati utilizzati in questi esempi, consulta Tipi di dati e regole di denominazione supportati in Amazon DynamoDB.

Scrittura di un elemento

Crea un modulo Node.js con il nome del file `put-item.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Crea un oggetto JSON che contiene i parametri necessari per aggiungere una voce, che in questo esempio include il nome della tabella e una mappa che definisce gli attributi da impostare e i valori per ogni attributo. Chiama il `PutItemCommand` metodo dell'oggetto del servizio client 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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Aggiornamento di un elemento

Crea un modulo Node.js con il nome del file `update-item.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Crea un oggetto JSON contenente i parametri necessari per aggiungere un elemento, che in questo esempio include il nome della tabella, la chiave da aggiornare e l'espressione di data che mappa i nuovi nomi degli attributi e i valori per ogni nuovo attributo. Chiama il `UpdateItemCommand` metodo dell'oggetto del servizio client DynamoDB.

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

const client = new DynamoDBClient({});

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

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

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Ottenimento di un elemento

Crea un modulo Node.js con il nome del file `get-item.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Per identificare la voce da ottenere, è necessario fornire il valore della chiave primaria per la voce nella tabella. Per impostazione predefinita, il metodo `GetItemCommand` restituisce tutti i valori degli attributi definiti per la voce. Per ottenere solo un sottoinsieme di tutti i possibili valori degli attributi, specifica un'espressione di proiezione.

Crea un oggetto JSON che contiene i parametri necessari per ottenere una voce, che in questo esempio include il nome della tabella, il nome e il valore della chiave per la voce che si cerca di ottenere e un'espressione di proiezione che identifica l'attributo della voce che si desidera recuperare. Chiama il `GetItemCommand` metodo dell'oggetto del servizio client DynamoDB.

Il seguente esempio di codice recupera un elemento da una tabella con una chiave primaria composta solo da una chiave di partizione e non da una chiave di partizione e non da una chiave di partizione e da una chiave di ordinamento. Se la tabella ha una chiave primaria composta da una chiave di partizione e una chiave di ordinamento, è necessario specificare anche il nome e l'attributo della chiave di ordinamento.

```
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;
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione di un elemento

Crea un modulo Node.js con il nome del file `delete-item.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Crea un oggetto JSON che contiene i parametri necessari per eliminare una voce, che in questo esempio include il nome della tabella e il nome e il valore della chiave della voce che si sta eliminando. Chiama il `DeleteItemCommand` metodo dell'oggetto del servizio client 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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Lettura e scrittura di elementi in batch in DynamoDB

Questo esempio di codice di Node.js illustra:

- Come leggere e scrivere batch di elementi in una tabella DynamoDB.

Lo scenario

In questo esempio, si utilizza una serie di moduli Node.js per inserire un batch di elementi in una tabella DynamoDB e leggere un batch di elementi. Il codice utilizza l'SDK per JavaScript eseguire operazioni di lettura e scrittura in batch utilizzando questi metodi della classe client DynamoDB:

- BatchGetItemCommand
- BatchWriteItemCommand

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB, vedere. Creazione e utilizzo di tabelle in DynamoDB

Important

Questi esempi utilizzano ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Note

Per informazioni sui tipi di dati utilizzati in questi esempi, consulta Tipi di dati e regole di denominazione supportati in Amazon DynamoDB.

Lettura di elementi in un batch

Crea un modulo Node.js con il nome del file `batch-get-item.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Crea un oggetto JSON che contiene i parametri necessari per ottenere un batch di voci, che in questo esempio include il nome di una o più tabelle da cui leggere, i valori delle chiavi da leggere in ciascuna tabella e l'espressione di proiezione che specifica gli attributi da restituire. Chiama il `BatchGetItemCommand` metodo dell'oggetto servizio 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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Scrittura di articoli in batch

Crea un modulo Node.js con il nome del file `batch-write-item.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Crea un oggetto JSON contenente i parametri necessari per ottenere un batch di elementi, che in questo esempio include la tabella in cui desideri scrivere gli elementi, le chiavi che desideri scrivere per ogni elemento e gli attributi insieme ai relativi valori. Chiama il `BatchWriteItemCommand` metodo dell'oggetto servizio DynamoDB.

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

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

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

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

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Interrogazione e scansione di una tabella DynamoDB

Questo esempio di codice di Node.js illustra:

- Come interrogare e scansionare una tabella DynamoDB alla ricerca di elementi.

Lo scenario

Eseguire le query consente di trovare le voci in una tabella o un indice secondario utilizzando solo i valori degli attributi della chiave primaria. È necessario fornire un nome e un valore della chiave di partizione da cercare. Puoi inoltre fornire un nome e un valore della chiave di ordinamento e utilizzare un operatore di confronto per perfezionare i risultati della ricerca. La scansione permette di trovare le voci controllando ogni voce nella tabella specificata.

In questo esempio, si utilizza una serie di moduli Node.js per identificare uno o più elementi che si desidera recuperare da una tabella DynamoDB. Il codice utilizza l'SDK per JavaScript interrogare e scansionare le tabelle utilizzando questi metodi della classe client DynamoDB:

- QueryCommand
- ScanCommand

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi esempi di Node.js e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB, vedere. Creazione e utilizzo di tabelle in DynamoDB

Important

Questi esempi utilizzano ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Note

Per informazioni sui tipi di dati utilizzati in questi esempi, consulta Tipi di dati e regole di denominazione supportati in Amazon DynamoDB.

Esecuzione di query su una tabella

Crea un modulo Node.js con il nome del file `query.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Crea un oggetto JSON che contiene i parametri necessari per eseguire una query sulla tabella, che in questo esempio include il nome della tabella, i `ExpressionAttributeValues` necessari per la query, una `KeyConditionExpression` che utilizza tali valori per definire quali voci la query restituisce e i nomi dei valori degli attributi da restituire per ciascuna voce. Chiama il `QueryCommand` metodo dell'oggetto servizio 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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node query.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Scansione di una tabella

Crea un modulo Node.js con il nome del file `scan.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea DynamoDB un oggetto servizio client. Crea un oggetto JSON che contiene i parametri necessari per eseguire la scansione della tabella per le voci, che in questo esempio include il nome della tabella, l'elenco dei valori degli attributi da restituire per ogni corrispondenza e un'espressione per filtrare il set di risultati per trovare le voci che contengono una determinata frase. Chiama il `ScanCommand` metodo dell'oggetto servizio 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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node scan.js
```

Questo codice di esempio è disponibile qui su GitHub.

Utilizzo del DynamoDB Document Client

Questo esempio di codice di Node.js illustra:

- Come accedere a una tabella DynamoDB utilizzando le utilità DynamoDB.

Lo scenario

Il DynamoDB Document Client semplifica l'utilizzo degli elementi astraendo la nozione di valori degli attributi. Questa astrazione annota i JavaScript tipi nativi forniti come parametri di input e converte i dati di risposta annotati in tipi nativi. JavaScript

Per ulteriori informazioni sul DynamoDB Document Client, vedere @aws -sdk/lib-dynamodb README on. GitHub Per ulteriori informazioni sulla programmazione con Amazon DynamoDB, consulta Programming with DynamoDB nella Amazon DynamoDB Developer Guide.

In questo esempio, si utilizza una serie di moduli Node.js per eseguire operazioni di base su una tabella DynamoDB utilizzando le utilità DynamoDB. Il codice utilizza l'SDK per interrogare e JavaScript scansionare le tabelle utilizzando questi metodi della classe DynamoDB Document Client:

- GetCommand
- PutCommand
- UpdateCommand
- QueryCommand
- DeleteCommand

Per ulteriori informazioni sulla configurazione del DynamoDB Document Client, vedere @aws -sdk/lib-dynamodb.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi esempi di Node.js e installa i moduli richiesti e di terze parti. AWS SDK for JavaScript Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB utilizzando l'SDK JavaScript per, vedere. Creazione e utilizzo di tabelle in DynamoDB Puoi anche utilizzare la console DynamoDB per creare una tabella.

> **Important**
>
> Questi esempi utilizzano ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .

Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

> **Note**
>
> Per informazioni sui tipi di dati utilizzati in questi esempi, consulta Tipi di dati e regole di denominazione supportati in Amazon DynamoDB.

Ottenere un item da una tabella.

Crea un modulo Node.js con il nome del file `get.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Ciò include `@aws-sdk/lib-dynamodb` un pacchetto di libreria che fornisce funzionalità client per documenti a`@aws-sdk/client-dynamodb`. Successivamente, imposta la configurazione come illustrato di seguito per il marshalling e il demarshalling, come secondo parametro opzionale, durante la creazione del client per documenti. Quindi, crea i client. Ora crea un oggetto JSON contenente i parametri necessari per ottenere un elemento dalla tabella, che in questo esempio include il nome della tabella, il nome della chiave hash in quella tabella e il valore della chiave hash per l'elemento che desideri ottenere. Chiama il `GetCommand` metodo del DynamoDB Document Client.

```
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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node get.js
```

Questo codice di esempio è disponibile qui su GitHub.

Inserimento di una voce in una tabella

Crea un modulo Node.js con il nome del file `put.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Ciò include `@aws-sdk/lib-dynamodb` un pacchetto di libreria che fornisce funzionalità client per documenti a`@aws-sdk/client-dynamodb`. Successivamente, imposta la configurazione come illustrato di seguito per il marshalling e il demarshalling, come secondo parametro opzionale, durante la creazione del client per documenti. Quindi, crea i client. Crea un oggetto JSON contenente i parametri necessari per scrivere un elemento nella tabella, che in questo esempio include il nome della tabella e una descrizione dell'elemento da aggiungere o aggiornare che includa la chiave hash e il valore e i nomi e i valori per gli attributi da impostare sull'elemento. Chiama il `PutCommand` metodo del DynamoDB Document Client.

```
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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node put.js
```

Questo codice di esempio è disponibile qui su GitHub.

Aggiornamento di una voce in una tabella

Crea un modulo Node.js con il nome del file `update.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Ciò include `@aws-sdk/lib-dynamodb` un pacchetto di libreria che fornisce funzionalità client per documenti a`@aws-sdk/client-dynamodb`. Successivamente, imposta la configurazione come illustrato di seguito per il marshalling e il demarshalling, come secondo parametro opzionale, durante la creazione del client per documenti. Quindi, crea i client. Crea un oggetto JSON contenente i parametri necessari per scrivere un elemento nella tabella, che in questo esempio include il nome della tabella, la chiave dell'elemento da aggiornare, un set `UpdateExpressions` che definisca gli attributi dell'elemento da aggiornare con i token a cui assegnare valori nei `ExpressionAttributeValues` parametri.Chiama il metodo `UpdateCommand` del DynamoDB Document Client.

```
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;
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node update.js
```

Questo codice di esempio è disponibile qui su GitHub.

Esecuzione di query su una tabella

Crea un modulo Node.js con il nome del file `query.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Ciò include `@aws-sdk/lib-dynamodb` un pacchetto di libreria che fornisce funzionalità client per documenti a`@aws-sdk/client-dynamodb`. Crea un oggetto JSON che contiene i parametri necessari per eseguire una query sulla tabella, che in questo esempio include il nome della tabella, i `ExpressionAttributeValues` necessari per la query e una `KeyConditionExpression` che utilizza tali valori per definire quali voci la query restituisce, Chiama il `QueryCommand` metodo del DynamoDB Document Client.

```
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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node query.js
```

Questo codice di esempio è disponibile qui su GitHub.

Eliminazione di una voce da una tabella.

Crea un modulo Node.js con il nome del file `delete.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Ciò include `@aws-sdk/lib-dynamodb` un pacchetto di libreria che fornisce funzionalità client per documenti a`@aws-sdk/client-dynamodb`. Successivamente, imposta la configurazione come illustrato di seguito per il marshalling e il demarshalling, come secondo parametro opzionale, durante la creazione del client per documenti. Quindi, crea i client. Per accedere a DynamoDB, crea un oggetto. `DynamoDB` Crea un oggetto JSON contenente i parametri necessari per eliminare un elemento nella tabella, che in questo esempio include il nome della tabella e il nome e il valore dell'hashkey dell'elemento che desideri eliminare. Chiama il `DeleteCommand` metodo del DynamoDB Document Client.

```
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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node delete.js
```

Questo codice di esempio è disponibile qui su GitHub.

Esempi di AWS Elemental MediaConvert

AWS Elemental MediaConvert è un servizio per la transcodifica di video basata su file con caratteristiche di trasmissione. Puoi usarlo per creare risorse per la trasmissione e la distribuzione video-on-demand (VOD) su Internet. Per ulteriori informazioni, consulta la Guida per l'utente di AWS Elemental MediaConvert.

L' JavaScript API for MediaConvert è esposta tramite la classe `MediaConvert` client. Per ulteriori informazioni, consulta Class: MediaConvert nel riferimento API.

Argomenti

- Ottenimento dell'endpoint specifico della regione per MediaConvert
- Creazione e gestione di lavori di transcodifica in MediaConvert
- Utilizzo dei modelli di lavoro in MediaConvert

Ottenimento dell'endpoint specifico della regione per MediaConvert

Questo esempio di codice di Node.js illustra:

- Come recuperare l'endpoint specifico della regione da. MediaConvert

Lo scenario

In questo esempio, si utilizza un modulo Node.js per chiamare MediaConvert e recuperare l'endpoint specifico della regione. È possibile recuperare l'URL dell'endpoint dall'endpoint predefinito del servizio e quindi non è ancora necessario l'endpoint specifico della regione. Il codice utilizza l'SDK per JavaScript recuperare questo endpoint utilizzando questo metodo della classe client: MediaConvert

- `DescribeEndpointsCommand`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.
- Crea un ruolo IAM che dia MediaConvert accesso ai tuoi file di input e ai bucket Amazon S3 in cui sono archiviati i file di output. Per i dettagli, consulta Configurare le autorizzazioni IAM nella Guida per l'AWS Elemental MediaConvertutente.

> **Important**
>
> Questo esempio utilizza ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Ottenere l'URL dell'endpoint

Crea una `libs` directory e crea un modulo Node.js con il nome `emcClientGet.js` del file. Copia e incolla il codice seguente al suo interno, che crea l'oggetto MediaConvert client. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `emc_getendpoint.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Create un oggetto per passare i parametri di richiesta vuoti per il `DescribeEndpointsCommand` metodo della classe MediaConvert client. Quindi chiama il metodo `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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node emc_getendpoint.js
```

Questo codice di esempio è disponibile qui su GitHub.

Creazione e gestione di lavori di transcodifica in MediaConvert

Questo esempio di codice di Node.js illustra:

- Come specificare l'endpoint specifico della regione con cui utilizzare. MediaConvert
- Come creare lavori di transcodifica in. MediaConvert
- Come annullare un processo di transcodifica.

- Come recuperare il file JSON per un processo di transcodifica completato.
- Come recuperare un array JSON per un massimo di 20 processi creati più di recente.

Lo scenario

In questo esempio, si utilizza un modulo Node.js per chiamare per MediaConvert creare e gestire lavori di transcodifica. A tale scopo, il codice utilizza l' JavaScript SDK utilizzando questi metodi della MediaConvert classe client:

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

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.
- Crea e configura bucket Amazon S3 che forniscono storage per i file di input e output dei job. Per i dettagli, consulta Creare spazio di archiviazione per i file nella Guida per l'AWS Elemental MediaConvertutente.
- Carica il video di input nel bucket Amazon S3 che hai fornito per lo storage di input. Per un elenco dei codec e contenitori di input video supportati, consulta Codec e contenitori di input supportati nella Guida per l'utente. AWS Elemental MediaConvert
- Crea un ruolo IAM che dia MediaConvert accesso ai tuoi file di input e ai bucket Amazon S3 in cui sono archiviati i file di output. Per i dettagli, consulta Configurare le autorizzazioni IAM nella Guida per l'AWS Elemental MediaConvertutente.

> **Important**
>
> Questo esempio utilizza ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Configurazione dell'SDK

Configura l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Poiché MediaConvert utilizza endpoint personalizzati per ogni account, devi anche configurare la classe `MediaConvert` client per utilizzare l'endpoint specifico della regione. A questo proposito, imposta il parametro `endpoint` su `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";
```

Definizione di un semplice processo di transcodifica

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `emcClient.js` Copia e incolla il codice seguente al suo interno, che crea l'oggetto MediaConvert client. Sostituisci *`REGION`* con la tua AWS regione. Sostituisci *`ENDPOINT`* con l'endpoint del tuo MediaConvert account, operazione che puoi fare nella pagina Account della MediaConvert console.

```
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 };
```

Questo codice di esempio può essere trovato qui su. GitHub

Crea un modulo Node.js con il nome del file `emc_createjob.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Crea il file JSON che definisce i parametri del processo di transcodifica.

Questi parametri sono dettagliati. È possibile utilizzare la AWS Elemental MediaConvertconsole per generare i parametri del lavoro JSON scegliendo le impostazioni del processo nella console e quindi selezionando Mostra lavoro JSON nella parte inferiore della sezione Job. Questo esempio illustra il JSON per un processo semplice.

> **Note**
>
> *Sostituisci JOB_QUEUE_ARN con la coda dei MediaConvert processi, IAM_ROLE_ARN con l'Amazon Resource Name (ARN) del ruolo IAM, OUTPUT_BUCKET_NAME con il nome del bucket di destinazione, ad esempio "s3://OUTPUT_BUCKET_NAME/ «, e INPUT_BUCKET_AND_FILENAME con il bucket di input e il nome del file, ad esempio" 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",
    },
  },
};
```

Creazione di un processo di transcodifica

Dopo aver creato i parametri di lavoro JSON, chiamate il `run` metodo asincrono per richiamare un oggetto del servizio `MediaConvert` client, passando i parametri. L'ID del processo creato viene restituito nei `data` della risposta.

```
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();
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node emc_createjob.js
```

Questo codice di esempio completo è disponibile qui su GitHub.

Annullamento di un processo di transcodifica

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `emcClient.js` Copia e incolla il codice seguente al suo interno, che crea l'oggetto MediaConvert client. Sostituisci *`REGION`* con la tua AWS regione. Sostituisci *`ENDPOINT`* con l'endpoint del tuo MediaConvert account, operazione che puoi fare nella pagina Account della MediaConvert console.

```
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 };
```

Questo codice di esempio può essere trovato qui su. GitHub

Crea un modulo Node.js con il nome del file `emc_canceljob.js`. Assicurati di configurare l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Crea il file JSON che include l'ID del processo da annullare. Quindi chiamate il `CancelJobCommand` metodo creando una promessa per richiamare un oggetto `MediaConvert` del servizio client, passando i parametri. Gestisci la risposta restituita dal callback della promessa.

> **Note**
>
> Sostituisci *`JOB_ID`* con l'ID del lavoro da annullare.

```
// 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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node ec2_canceljob.js
```

Questo codice di esempio è disponibile qui su GitHub.

Elenco dei lavori di transcodifica recenti

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `emcClient.js` Copia e incolla il codice seguente al suo interno, che crea l'oggetto MediaConvert client. Sostituisci *`REGION`* con la tua AWS regione. Sostituisci *`ENDPOINT`* con l'endpoint del tuo MediaConvert account, operazione che puoi fare nella pagina Account della MediaConvert console.

```
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 };
```

Questo codice di esempio può essere trovato qui su. GitHub

Crea un modulo Node.js con il nome del file `emc_listjobs.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea i parametri JSON, inclusi i valori per specificare se ordinare o `DESCENDING` ordinare l'elenco in `ASCENDING` base all'Amazon Resource Name (ARN) della coda dei lavori da controllare e lo stato dei lavori da includere. Quindi chiama il `ListJobsCommand` metodo creando una promessa per richiamare un oggetto `MediaConvert` del servizio client, passando i parametri.

> **Note**
>
> Sostituisci *QUEUE_ARN* con l'Amazon Resource Name (ARN) della coda dei lavori da controllare e *STATUS* con lo stato della coda.

```
// 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();
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node emc_listjobs.js
```

Questo codice di esempio è disponibile qui su GitHub.

Utilizzo dei modelli di lavoro in MediaConvert

Questo esempio di codice di Node.js illustra:

- Come creare modelli dei processi AWS Elemental MediaConvert.
- Come utilizzare un modello dei processi per creare un processo di transcodifica.
- Come elencare tutti i modelli dei processi.
- Come eliminare i modelli dei processi

Lo scenario

Il codice JSON richiesto per creare un processo di transcodifica in MediaConvert è dettagliato e contiene un gran numero di impostazioni. È possibile semplificare notevolmente la creazione del processo salvando le impostazioni corrette in un modello del processo che è possibile utilizzare per creare processi successivi. In questo esempio, si utilizza un modulo Node.js per chiamare per MediaConvert creare, utilizzare e gestire modelli di lavoro. A tale scopo, il codice utilizza l'SDK utilizzando questi metodi della classe MediaConvert client: JavaScript

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

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.
- Crea un ruolo IAM che dia MediaConvert accesso ai tuoi file di input e ai bucket Amazon S3 in cui sono archiviati i file di output. Per i dettagli, consulta Configurare le autorizzazioni IAM nella Guida per l'AWS Elemental MediaConvertutente.

> **Important**
>
> Questi esempi utilizzano ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Creazione di un modello di lavoro

Crea una `libs` directory e crea un modulo Node.js con il nome del file`emcClient.js`. Copia e incolla il codice seguente al suo interno, che crea l'oggetto MediaConvert client. Sostituisci *`REGION`* con la tua AWS regione. Sostituisci *`ENDPOINT`* con l'endpoint del tuo MediaConvert account, operazione che puoi fare nella pagina Account della MediaConvert console.

```
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 };
```

Questo codice di esempio può essere trovato qui su. GitHub

Crea un modulo Node.js con il nome del file `emc_create_jobtemplate.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Specifica il file JSON dei parametri per la creazione del modello. È possibile utilizzare la maggior parte dei parametri JSON da un processo di successo precedente per specificare i valori `Settings` nel modello. In questo esempio vengono utilizzate le impostazioni del processo contenute in Creazione e gestione di lavori di transcodifica in MediaConvert.

Chiamate il `CreateJobTemplateCommand` metodo creando una promessa per richiamare un oggetto `MediaConvert` del servizio client, passando i parametri.

> **Note**
>
> Sostituisci *JOB_QUEUE_ARN* con l'Amazon Resource Name (ARN) della coda dei lavori da controllare e *BUCKET_NAME con il nome* del bucket Amazon S3 di destinazione, ad esempio "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();
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node emc_create_jobtemplate.js
```

Questo codice di esempio è disponibile qui su GitHub.

Creazione di un processo di transcodifica da un modello di lavoro

Create una `libs` directory e create un modulo Node.js con il nome del file. `emcClient.js` Copia e incolla il codice seguente al suo interno, che crea l'oggetto MediaConvert client. Sostituisci *`REGION`* con la tua AWS regione. Sostituisci *`ENDPOINT`* con l'endpoint del tuo MediaConvert account, operazione che puoi fare nella pagina Account della MediaConvert console.

```
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 };
```

Questo codice di esempio può essere trovato qui su. GitHub

Crea un modulo Node.js con il nome del file `emc_template_createjob.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea il JSON dei parametri di creazione del processo, tra cui il nome del modello del processo e le `Settings` da utilizzare che sono specifiche per il processo. Quindi chiamate il `CreateJobsCommand` metodo creando una promessa per richiamare un oggetto `MediaConvert` del servizio client, passando i parametri.

> **Note**
>
> *Sostituisci JOB_QUEUE_ARN con l'Amazon Resource Name (ARN) della coda dei lavori da controllare, KEY_PAIR_NAME con, TEMPLATE_NAME con, ROLE_ARN con l'Amazon Resource Name (ARN) del ruolo e INPUT_BUCKET_AND_FILENAME con il bucket di input e il nome del file, ad esempio "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();
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node emc_template_createjob.js
```

Questo codice di esempio è disponibile qui su GitHub.

Elencare i modelli di lavoro

Crea una `libs` directory e crea un modulo Node.js con il nome del file`emcClient.js`. Copia e incolla il codice seguente al suo interno, che crea l'oggetto MediaConvert client. Sostituisci *`REGION`* con la tua AWS regione. Sostituisci *`ENDPOINT`* con l'endpoint del tuo MediaConvert account, operazione che puoi fare nella pagina Account della MediaConvert console.

```
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 };
```

Questo codice di esempio può essere trovato qui su. GitHub

Crea un modulo Node.js con il nome del file `emc_listtemplates.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire i parametri della richiesta vuoti per il metodo `listTemplates` della classe del client `MediaConvert`. Includi valori per determinare i modelli da elencare (`NAME`, `CREATION DATE`, `SYSTEM`), il numero da elencare e il loro ordinamento. Per chiamare il `ListTemplatesCommand` metodo, create una promessa di richiamo di un oggetto MediaConvert del servizio client, passando i parametri.

```
// 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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node emc_listtemplates.js
```

Questo codice di esempio è disponibile qui su GitHub.

Eliminazione di un modello di lavoro

Crea una `libs` directory e crea un modulo Node.js con il nome `emcClient.js` del file. Copia e incolla il codice seguente al suo interno, che crea l'oggetto MediaConvert client. Sostituisci *`REGION`* con la tua AWS regione. Sostituisci *`ENDPOINT`* con l'endpoint del tuo MediaConvert account, operazione che puoi fare nella pagina Account della MediaConvert console.

```
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 };
```

Questo codice di esempio può essere trovato qui su. GitHub

Crea un modulo Node.js con il nome del file `emc_deletetemplate.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per inoltrare il nome del modello del processo da eliminare come parametri per il metodo `DeleteJobTemplateCommand` della classe client `MediaConvert`. Per chiamare il `DeleteJobTemplateCommand` metodo, create una promessa di richiamo di un oggetto MediaConvert del servizio client, passando i parametri.

```
// 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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node emc_deletetemplate.js
```

Questo codice di esempio è disponibile qui su GitHub.

Esempi di AWS Lambda

AWS Lambdaè un servizio di elaborazione senza server che consente di eseguire codice senza fornire o gestire server, creare una logica di scalabilità dei cluster sensibile al carico di lavoro, mantenere integrazioni di eventi o gestire i runtime.

L'JavaScriptAPI per AWS Lambda è esposta tramite la classe LambdaServiceclient.

Ecco un elenco di esempi che dimostrano come creare e utilizzare le funzioni Lambda con la AWS SDK for JavaScript v3:

- Richiamare Lambda con API Gateway
- Creazione di eventi pianificati per eseguire AWS Lambda funzioni

Esempi di Amazon Lex

Amazon Lex è un AWS servizio per la creazione di interfacce conversazionali in applicazioni che utilizzano voce e testo.

L'JavaScriptAPI per Amazon Lex è esposta tramite la classe client Lex Runtime Service.

- Creazione di un chatbot Amazon Lex

Esempi di Amazon Polly

Questo esempio di codice di Node.js illustra:

- Caricare l'audio registrato con Amazon Polly su Amazon S3

Lo scenario

In questo esempio, una serie di moduli Node.js viene utilizzata per caricare automaticamente l'audio registrato utilizzando Amazon Polly su Amazon S3 utilizzando questi metodi della classe client Amazon S3:

- `StartSpeechSynthesisTaskCommand`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura un ambiente di progetto per eseguire JavaScript esempi di Node seguendo le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.
- Crea un ruolo utente Amazon Cognito AWS Identity and Access Management (IAM) non autenticato pollySynthesizeSpeech : permessi e un pool di identità Amazon Cognito a cui è associato il ruolo IAM. La Crea le risorse utilizzando il AWSAWS CloudFormation sezione seguente descrive come creare queste risorse.

> **Note**
>
> Questo esempio utilizza Amazon Cognito, ma se non utilizzi Amazon Cognito, l'utente deve avere AWS la seguente politica di autorizzazione IAM

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

Crea le risorse utilizzando il AWSAWS CloudFormation

AWS CloudFormation consente di creare ed effettuare il provisioning delle distribuzioni dell'infrastruttura AWS in modo ripetuto e prevedibile. Per ulteriori informazioni su AWS CloudFormation, consulta la Guida per l'utente di AWS CloudFormation.

Per creare lo AWS CloudFormation stack:

1. Installa e configura le AWS CLI seguenti istruzioni contenute nella Guida per l'AWS CLIutente.
2. Create un file denominato `setup.yaml` nella directory principale della cartella del progetto e copiatene il contenuto. GitHub

> **Note**
>
> Il AWS CloudFormation modello è stato generato utilizzando quello AWS CDK disponibile qui GitHub. Per ulteriori informazioni su AWS CDK, consulta la Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK).

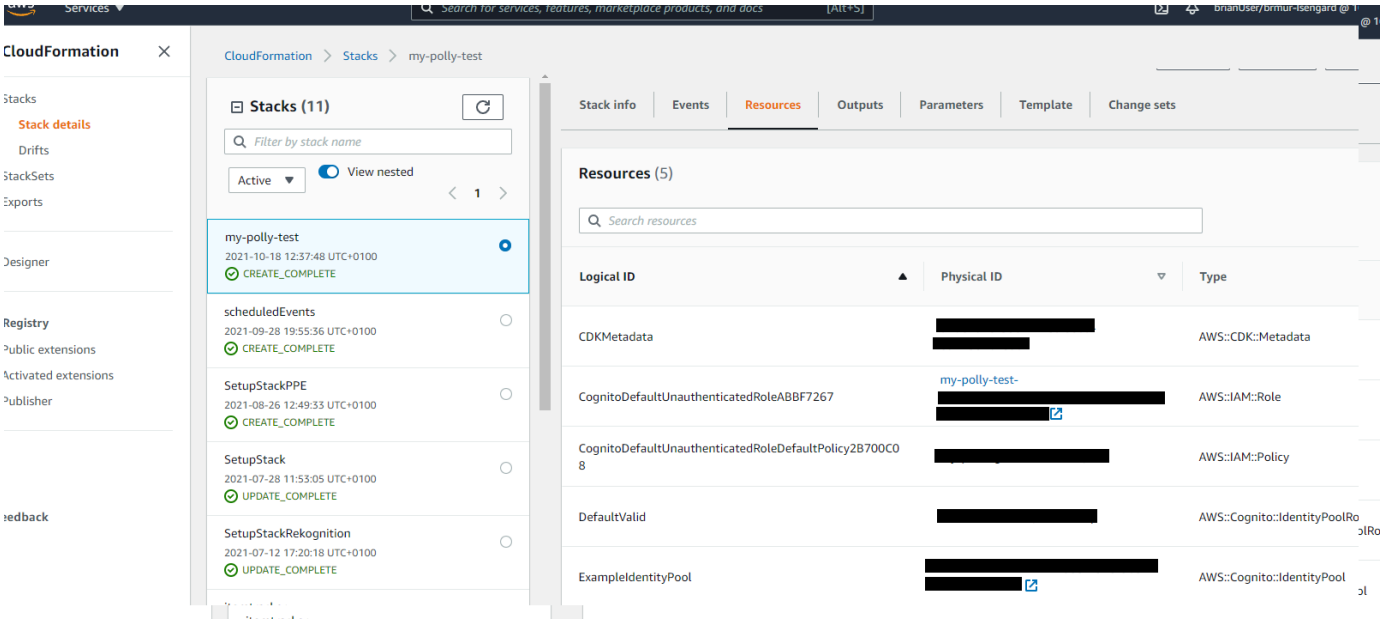
3. Esegui il seguente comando dalla riga di comando, sostituendo *`STACK_NAME con un nome`* univoco per lo stack.

> **Important**
>
> Il nome dello stack deve essere univoco all'interno di una regione e di un account. AWS AWS È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

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

Per ulteriori informazioni sui parametri dei `create-stack` comandi, consultate la guida di riferimento ai AWS CLI comandi e la Guida per l'AWS CloudFormationutente.

4. Accedi alla console di AWS CloudFormation gestione, scegli Stack, scegli il nome dello stack e scegli la scheda Risorse per visualizzare un elenco delle risorse create.



Caricare l'audio registrato con Amazon Polly su Amazon S3

Crea un modulo Node.js con il nome del file `polly_synthesize_to_s3.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Nel codice, inserisci *`REGION`* e *`BUCKET_NAME`*. Per accedere ad Amazon Polly, crea un oggetto di servizio `Polly` client. Sostituisci *`«IDENTITY_POOL_ID»`* con quello `IdentityPoolId`

dalla pagina di esempio del pool di identità di Amazon Cognito che hai creato per questo esempio. Questo viene passato anche a ciascun oggetto client.

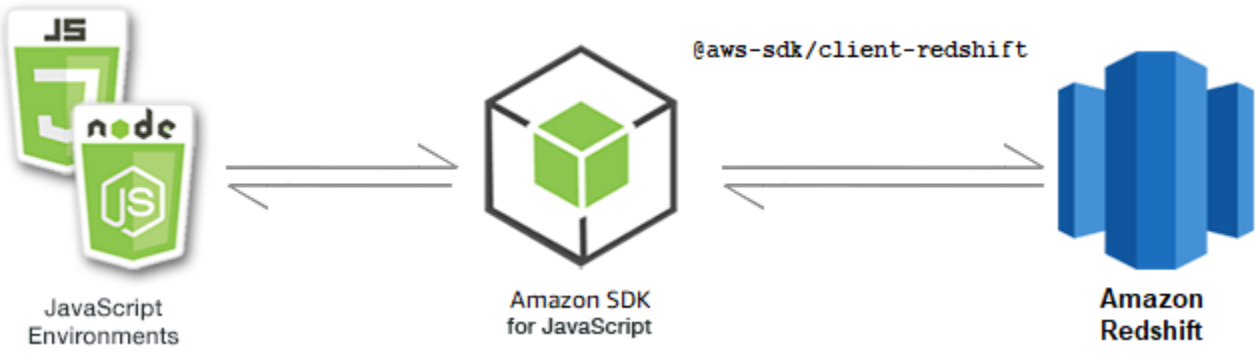
Chiama il `StartSpeechSynthesisCommand` metodo dell'oggetto del servizio client Amazon Polly, sintetizza il messaggio vocale e caricalo nel bucket Amazon S3.

```
const { StartSpeechSynthesisTaskCommand } = require("@aws-sdk/client-polly");
const { pollyClient } = require("./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();
```

Questo codice di esempio può essere trovato qui su. GitHub

Esempi di Amazon Redshift

Amazon Redshift è un servizio di data warehouse nel cloud in scala petabyte interamente gestito. Un data warehouse Amazon Redshift è costituito da un insieme di risorse di calcolo denominate nodi, strutturate in un gruppo denominato cluster. Ciascun cluster esegue un motore Amazon Redshift e contiene uno o più database.



L' JavaScript API per Amazon Redshift è esposta tramite la classe client Amazon Redshift.

Argomenti

- Esempi di Amazon Redshift

Esempi di Amazon Redshift

In questo esempio, una serie di moduli Node.js vengono utilizzati per creare, modificare, descrivere i parametri e quindi eliminare i cluster Amazon Redshift utilizzando i seguenti metodi della `Redshift` classe client:

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

Per ulteriori informazioni sugli utenti di Amazon Redshift, consulta la guida introduttiva di Amazon Redshift.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi JavaScript Sintassi ES6/CommonJS

Creazione di un cluster Amazon Redshift

Questo esempio dimostra come creare un cluster Amazon Redshift utilizzando. AWS SDK for JavaScript Per ulteriori informazioni, consulta. CreateCluster

> **Important**
>
> Il cluster che state per creare è attivo (e non è in esecuzione in una sandbox). Saranno addebitati i costi standard di utilizzo di Amazon Redshift relativi al cluster finché non viene eliminato. Se elimini il cluster nella stessa sessione in cui lo hai creato, i costi totali sono minimi.

Create una `libs` directory e create un modulo Node.js con il nome del file`redshiftClient.js`. Copia e incolla il codice seguente al suo interno, che crea l'oggetto client Amazon Redshift. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `redshift-create-cluster.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Create un oggetto parametrico, specificando il tipo di nodo da assegnare e le credenziali

di accesso principali per l'istanza di database creata automaticamente nel cluster e infine il tipo di cluster.

> **Note**
>
> Sostituisci *CLUSTER_NAME con il nome del* cluster. Per *NODE_TYPE*, specificate il tipo di nodo da fornire, ad esempio 'dc2.large'. *MASTER_USER_USER_PASSWORD sono le credenziali di accesso dell'utente master* dell'istanza DB nel cluster. *Per CLUSTER_TYPE, inserisci il tipo di cluster.* Se lo specificate`single-node`, non è necessario il parametro. `NumberOfNodes` I parametri rimanenti sono opzionali.

```
// 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();
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui. GitHub

Modifica di un cluster Amazon Redshift

Questo esempio mostra come modificare la password dell'utente principale di un cluster Amazon Redshift utilizzando. AWS SDK for JavaScript Per ulteriori informazioni su quali altre impostazioni puoi modificare, consulta ModifyCluster.

Create una `libs` directory e create un modulo Node.js con il nome del file`redshiftClient.js`. Copia e incolla il codice seguente al suo interno, che crea l'oggetto client Amazon Redshift. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `redshift-modify-cluster.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Specificate la AWS regione, il nome del cluster che desiderate modificare e la nuova password dell'utente principale.

> **Note**
>
> Sostituite *`CLUSTER_NAME`* con il nome del cluster e *`MASTER_USER_PASSWORD con la nuova password dell'utente principale`*.

```
// 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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui. GitHub

Visualizzazione dei dettagli di un cluster Amazon Redshift

Questo esempio mostra come visualizzare i dettagli di un cluster Amazon Redshift utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni su optional, consulta DescribeClusters.

Create una `libs` directory e create un modulo Node.js con il nome del file`redshiftClient.js`. Copia e incolla il codice seguente al suo interno, che crea l'oggetto client Amazon Redshift. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `redshift-describe-clusters.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Specificate la AWS regione, il nome del cluster che desiderate modificare e la nuova password dell'utente principale.

> **Note**
>
> Sostituisci *CLUSTER_NAME* con il nome del cluster.

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

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

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

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui. GitHub

Eliminare un cluster Amazon Redshift

Questo esempio mostra come visualizzare i dettagli di un cluster Amazon Redshift utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni su quali altre impostazioni puoi modificare, consulta DeleteCluster.

Create una `libs` directory e create un modulo Node.js con il nome del file`redshiftClient.js`. Copia e incolla il codice seguente al suo interno, che crea l'oggetto client Amazon Redshift. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file`redshift-delete-clusters.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Specificate la AWS regione, il nome del cluster che desiderate modificare e la nuova password dell'utente principale. Specificare se si desidera salvare un'istantanea finale del cluster prima dell'eliminazione e, in tal caso, l'ID dell'istantanea.

> **Note**
>
> Sostituisci *`CLUSTER_NAME con il nome del`* cluster. Per il *`SkipFinalClusterSnapshot`*, specificare se creare un'istantanea finale del cluster prima di eliminarlo. *`Se specificate 'false', specificate l'id dell'istantanea finale del cluster in CLUSTER_SNAPSHOT_ID.`* È possibile ottenere questo ID facendo clic sul collegamento nella colonna Istantanee per il cluster nella dashboard dei cluster e scorrendo verso il basso fino al riquadro Snapshot. Nota che lo stem non `rs:` fa parte dell'ID dell'istantanea.

```
// 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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui. GitHub

Esempi di Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) Simple Email Service (Amazon SES) è un servizio di invio e-mail basato sul cloud progettato per aiutare i professionisti del marketing digitale e gli sviluppatori di applicazioni a inviare e-mail di marketing, notifiche e transazionali. Si tratta di un servizio affidabile, a costo ridotto per aziende di tutte le dimensioni che utilizzano la posta elettronica per mantenere il contatto con i clienti.

L' JavaScript API per Amazon SES è esposta tramite la classe SES client. Per ulteriori informazioni sull'uso della classe client Amazon SES, consulta Class: SES nell'API Reference.

Argomenti

- Gestione delle identità Amazon SES
- Utilizzo dei modelli di e-mail in Amazon SES
- Invio di e-mail tramite Amazon SES

Gestione delle identità Amazon SES

Questo esempio di codice di Node.js illustra:

- Come verificare gli indirizzi e-mail e i domini utilizzati con Amazon SES.
- Come assegnare una policy AWS Identity and Access Management (IAM) alle identità Amazon SES.
- Come elencare tutte le identità Amazon SES per il tuo AWS account.
- Come eliminare le identità utilizzate con Amazon SES.

Un'identità Amazon SES è un indirizzo e-mail o un dominio che Amazon SES utilizza per inviare e-mail. Amazon SES richiede che verifichi le tue identità e-mail, confermando che le possiedi e impedendo ad altri di utilizzarle.

Per dettagli su come verificare indirizzi e-mail e domini in Amazon SES, consulta la sezione Verifica degli indirizzi e-mail e dei domini in Amazon SES nella Amazon Simple Email Service Developer Guide. Per informazioni sull'autorizzazione all'invio in Amazon SES, consulta Panoramica dell'autorizzazione all'invio di Amazon SES.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per verificare e gestire le identità di Amazon SES. I moduli Node.js utilizzano l'SDK per JavaScript verificare indirizzi e-mail e domini, utilizzando questi metodi della SES classe client:

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

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWS SDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi. JavaScript Sintassi ES6/CommonJS

Elencare le tue identità

In questo esempio, utilizza un modulo Node.js per elencare gli indirizzi e-mail e i domini da utilizzare con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome `sesClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_listidentities.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire `IdentityType` e altri parametri per il metodo `ListIdentitiesCommand` della classe client SES. Per chiamare il `ListIdentitiesCommand` metodo, richiama un oggetto di servizio Amazon SES, passando l'oggetto parameters.

Il `data` valore restituito contiene una matrice di identità di dominio come specificato dal `IdentityType` parametro.

> **Note**
>
> Sostituisci *`IDENTITY_TYPE`* con il tipo di identità, che può essere "EmailAddress" o «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;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node ses_listidentities.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Verifica di un'identità indirizzo e-mail

In questo esempio, utilizza un modulo Node.js per verificare i mittenti di posta elettronica da utilizzare con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome `sesClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_verifyemailidentity.js`. Configura l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire il parametro `EmailAddress` per il metodo `VerifyEmailIdentityCommand` della classe client `SES`. Per chiamare il `VerifyEmailIdentityCommand` metodo, richiama un oggetto del servizio client Amazon SES, passando i parametri.

> **Note**
>
> Sostituisci *`ADDRESS@DOMAIN.EXT`* con l'indirizzo e-mail, ad esempio 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;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi. Il dominio viene aggiunto ad Amazon SES per essere verificato.

```
node ses_verifyemailidentity.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Verifica dell'identità di un dominio

In questo esempio, utilizza un modulo Node.js per verificare i domini e-mail da utilizzare con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome `sesClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_verifydomainidentity.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire il parametro `Domain` per il metodo `VerifyDomainIdentityCommand` della classe client `SES`. Per chiamare il `VerifyDomainIdentityCommand` metodo, richiama un oggetto del servizio client Amazon SES, passando l'oggetto parameters.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> Sostituisci *`AMI_ID`* con l'ID dell'Amazon Machine Image (AMI) da eseguire e *`KEY_PAIR_NAME della coppia di`* chiavi da assegnare all'ID 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;
  }
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi. Il dominio viene aggiunto ad Amazon SES per essere verificato.

```
node ses_verifydomainidentity.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione delle identità

In questo esempio, utilizza un modulo Node.js per eliminare gli indirizzi e-mail o i domini utilizzati con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome `sesClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_deleteidentity.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire il parametro `Identity` per il metodo `DeleteIdentityCommand` della classe client `SES`. Per chiamare il `DeleteIdentityCommand` metodo, crea un oggetto del

servizio client Amazon SES `request` per richiamare un oggetto del servizio client Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> Sostituisci *`IDENTITY_TYPE`* con il tipo di identità da eliminare e *`IDENTITY_NAME con il nome`* dell'identità da eliminare.

```
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;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node ses_deleteidentity.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Utilizzo dei modelli di e-mail in Amazon SES

Questo esempio di codice di Node.js illustra:

- Come ottenere un elenco di tutti i tuoi modelli di email.
- Come recuperare e aggiornare i modelli di email.
- Come creare ed eliminare modelli di email.

Amazon SES consente di inviare messaggi e-mail personalizzati utilizzando modelli di e-mail. Per dettagli su come creare e utilizzare modelli di e-mail in Amazon SES, consulta Invio di e-mail personalizzate utilizzando l'API Amazon SES nella Amazon Simple Email Service Developer Guide.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per utilizzare i modelli di e-mail. I moduli Node.js utilizzano l'SDK per JavaScript creare e utilizzare modelli di posta elettronica utilizzando questi metodi della classe SES client:

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

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWS SDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi. JavaScript Sintassi ES6/CommonJS

Elencare i tuoi modelli di email

In questo esempio, utilizza un modulo Node.js per creare un modello di e-mail da utilizzare con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`sesClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_listtemplates.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire i parametri per il metodo `ListTemplatesCommand` della classe client `SES`. Per chiamare il `ListTemplatesCommand` metodo, richiama un oggetto del servizio client Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> Sostituisci *`ITEMS_COUNT`* con il numero massimo di modelli da restituire. Il valore deve essere compreso tra un minimo di 1 e un massimo di 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;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi. Amazon SES restituisce l'elenco dei modelli.

```
node ses_listtemplates.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Ottenere un modello di email

In questo esempio, utilizza un modulo Node.js per ottenere un modello di e-mail da utilizzare con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`sesClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_gettemplate.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire il parametro `TemplateName` per il metodo `GetTemplateCommand` della classe client `SES`. Per chiamare il `GetTemplateCommand` metodo, richiama un oggetto del servizio client Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> Sostituisci *`TEMPLATE_NAME con il nome`* del modello da restituire.

```
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 (err) {
    console.log("Failed to get email template.", err);
    return err;
  }
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi. Amazon SES restituisce i dettagli del modello.

```
node ses_gettemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Creazione di un modello di email

In questo esempio, utilizza un modulo Node.js per creare un modello di e-mail da utilizzare con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`sesClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_createtemplate.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire i parametri per il metodo `CreateTemplateCommand` della classe client `SES`, inclusi `TemplateName`, `HtmlPart`, `SubjectPart` e `TextPart`. Per chiamare il `CreateTemplateCommand` metodo, richiama un oggetto del servizio client Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> *Sostituisci TEMPLATE_NAME con un nome per il nuovo modello, HTML_CONTENT con il contenuto dell'e-mail con tag HTML, SUBJECT con l'oggetto dell'e-mail e TEXT_CONTENT con il testo dell'e-mail.*

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
```

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

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi. Il modello viene aggiunto ad Amazon SES.

```
node ses_createtemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Aggiornamento di un modello di e-mail

In questo esempio, utilizza un modulo Node.js per creare un modello di e-mail da utilizzare con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`sesClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_updatetemplate.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire i valori dei parametri `Template` che desideri aggiornare nel modello, con il parametro `TemplateName` richiesto trasferito al metodo `UpdateTemplateCommand` della classe client `SES`. Per chiamare il `UpdateTemplateCommand` metodo, richiama un oggetto di servizio Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> *`Sostituisci TEMPLATE_NAME con un nome del modello, HTML_CONTENT con il contenuto dell'e-mail con tag HTML, SUBJECT con l'oggetto dell'e-mail e TEXT_CONTENT con il testo dell'e-mail.`*

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

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi. Amazon SES restituisce i dettagli del modello.

```
node ses_updatetemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione di un modello di e-mail

In questo esempio, utilizza un modulo Node.js per creare un modello di e-mail da utilizzare con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`sesClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_deletetemplate.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire il parametro `TemplateName` richiesto al metodo `DeleteTemplateCommand` della classe client `SES`. Per chiamare il `DeleteTemplateCommand` metodo, richiama un oggetto di servizio Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> Sostituisci *`TEMPLATE_NAME`* con il nome del modello da eliminare.

```
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;
  }
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi. Amazon SES restituisce i dettagli del modello.

```
node ses_deletetemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Invio di e-mail tramite Amazon SES

Questo esempio di codice di Node.js illustra:

- Come inviare un'e-mail di testo o in formato HTML.
- Come inviare e-mail in base a un modello di e-mail.
- Come inviare e-mail in blocco in base a un modello di e-mail.

L'API Amazon SES offre due modi diversi per inviare un'e-mail, a seconda del livello di controllo che desideri sulla composizione del messaggio e-mail: formattato e non elaborato. Per i dettagli, consulta Invio di e-mail formattate utilizzando l'API Amazon SES e Invio di e-mail non elaborate utilizzando l'API Amazon SES.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per inviare e-mail in diversi modi. I moduli Node.js utilizzano l'SDK per JavaScript creare e utilizzare modelli di posta elettronica utilizzando questi metodi della classe SES client:

- `SendEmailCommand`
- `SendTemplatedEmailCommand`
- `SendBulkTemplatedEmailCommand`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWS SDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi. JavaScript Sintassi ES6/CommonJS

Requisiti per l'invio di messaggi e-mail

Amazon SES compone un messaggio e-mail e lo mette immediatamente in coda per l'invio. Per inviare e-mail utilizzando il metodo `SendEmailCommand`, il messaggio deve soddisfare i seguenti requisiti:

- Devi inviare il messaggio da un dominio o da un indirizzo e-mail verificato. Se tenti di inviare e-mail utilizzando un dominio o un indirizzo non verificato, l'operazione genera un errore `"Email address not verified"`.
- Se il tuo account si trova ancora nella sandbox di Amazon SES, puoi inviare messaggi solo a domini o indirizzi verificati oppure a indirizzi e-mail associati al simulatore di mailbox di Amazon SES. Per ulteriori informazioni, consulta la sezione Verifica degli indirizzi e-mail e dei domini nella Amazon Simple Email Service Developer Guide.
- La dimensione totale del messaggio, inclusi gli allegati, deve essere inferiore a 10 MB.
- Il messaggio deve includere almeno l'indirizzo e-mail di un destinatario. L'indirizzo del destinatario può essere un indirizzo "To:" ("A:"), "CC:" ("Cc:") o "BCC:" ("Ccn:"). Se l'indirizzo e-mail del destinatario non è valido (ovvero non è nel formato`UserName@[SubDomain.]Domain.TopLevelDomain`), l'intero messaggio viene rifiutato, anche se contiene altri destinatari validi.
- Il messaggio non può includere più di 50 destinatari nei campi To:, CC: e BCC:. Se devi inviare un messaggio e-mail a un pubblico più ampio, puoi dividere l'elenco dei destinatari in gruppi di massimo 50 persone e quindi chiamare il metodo `sendEmail` più volte per inviare il messaggio a ciascun gruppo.

Invio di un'e-mail

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`sesClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_sendemail.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Create un oggetto per passare i valori dei parametri che definiscono l'e-mail da inviare, inclusi gli indirizzi del mittente e del destinatario, l'oggetto e il corpo dell'e-mail in formato testo semplice e HTML, al `SendEmailCommand` metodo della classe `SES` client. Per chiamare il `SendEmailCommand` metodo, richiama un oggetto di servizio Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> Sostituisci *`RECEIVER_ADDRESS`* con l'indirizzo a cui inviare l'e-mail e *`SENDER_ADDRESS con l'indirizzo`* e-mail da cui inviare l'e-mail.

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

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
```

```
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (e) {
    console.error("Failed to send email.");
    return e;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi. L'e-mail viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendemail.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Invio di un'e-mail utilizzando un modello

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES. Crea un modulo Node.js con il nome del file `ses_sendtemplatedemail.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire i valori dei parametri che definiscono l'e-mail da inviare, inclusi gli indirizzi dei mittenti e dei destinatari, l'oggetto, il corpo dell'e-mail in testo normale e in formato HTML, al metodo `SendTemplatedEmailCommand` della classe client `SES`. Per chiamare il `SendTemplatedEmailCommand` metodo, richiama un oggetto del servizio client Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> *Sostituisci REGION con la tua AWS regione, RECEIVER_ADDRESS con l'indirizzo a cui inviare l'e-mail, SENDER_ADDRESS con l'indirizzo e-mail da cui inviare l'e-mail e TEMPLATE_NAME con il nome del modello.*

```
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 (err) {
    console.log("Failed to send template email", err);
    return err;
  }
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi. L'e-mail viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendtemplatedemail.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Invio di e-mail in blocco utilizzando un modello

In questo esempio, utilizza un modulo Node.js per inviare e-mail con Amazon SES.

Crea una `libs` directory e crea un modulo Node.js con il nome `sesClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SES. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `ses_sendbulktemplatedemail.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Create un oggetto per passare i valori dei parametri che definiscono l'e-mail da inviare, inclusi gli indirizzi del mittente e del destinatario, l'oggetto e il corpo dell'e-mail in formato testo semplice e HTML, al `SendBulkTemplatedEmailCommand` metodo della classe `SES` client. Per chiamare il `SendBulkTemplatedEmailCommand` metodo, richiama un oggetto di servizio Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, vedi Utilizzo dei comandi V3.

> **Note**
>
> Sostituisci *RECEIVER_ADDRESSES* con l'indirizzo a cui inviare l'e-mail e *SENDER_ADDRESS con l'indirizzo* e-mail da cui inviare l'e-mail.

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

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map each
 user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
```

```
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (err) {
    console.log("Failed to send bulk template email", err);
    return err;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi. L'e-mail viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Esempi di servizi di notifica Amazon Simple

Amazon Simple Notification Service (Amazon SNS) è un servizio web che coordina e gestisce la consegna o l'invio di messaggi agli endpoint o ai clienti abbonati.

In Amazon SNS, esistono due tipi di clienti, editori e abbonati, noti anche come produttori e consumatori.

Gli editori comunicano in modo asincrono con i sottoscrittori producendo e inviando un messaggio a un argomento, che rappresenta un punto di accesso logico e un canale di comunicazione. Gli abbonati (server Web, indirizzi e-mail, code Amazon SQSAWS Lambda, funzioni) utilizzano o ricevono il messaggio o la notifica tramite uno dei protocolli supportati (Amazon SQS, HTTP/S, e-mailAWS Lambda, SMS) quando sono abbonati all'argomento.

L' JavaScript API per Amazon SNS è esposta tramite la classe: SNS.

Argomenti

- Gestione degli argomenti in Amazon SNS
- Pubblicazione di messaggi in Amazon SNS
- Gestione degli abbonamenti in Amazon SNS
- Invio di messaggi SMS con Amazon SNS

Gestione degli argomenti in Amazon SNS

Questo esempio di codice di Node.js illustra:

- Come creare argomenti in Amazon SNS su cui pubblicare notifiche.
- Come eliminare argomenti creati in Amazon SNS.
- Come ottenere un elenco degli argomenti disponibili.

- Come ottenere e impostare gli attributi di argomento.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per creare, elencare ed eliminare argomenti di Amazon SNS e per gestire gli attributi degli argomenti. I moduli Node.js utilizzano l'SDK per JavaScript gestire gli argomenti utilizzando questi metodi della classe SNS client:

- `CreateTopicCommand`
- `ListTopicsCommand`
- `DeleteTopicCommand`
- `GetTopicAttributesCommand`
- `SetTopicAttributesCommand`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi. JavaScript Sintassi ES6/CommonJS

Creazione di un argomento

In questo esempio, usa un modulo Node.js per creare un argomento Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome `snsClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `create-topic.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto per trasferire l'oggetto `Name` per il nuovo argomento al metodo `CreateTopicCommand` della classe client SNS. Per chiamare il `CreateTopicCommand` metodo, crea una funzione asincrona che richiama un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Il `data` valore restituito contiene l'ARN dell'argomento.

> **Note**
>
> Sostituisci *`TOPIC_NAME con il nome`* dell'argomento.

```
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;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Elenco dei tuoi argomenti

In questo esempio, usa un modulo Node.js per elencare tutti gli argomenti di Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome `snsClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `list-topics.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto vuoto da trasferire al metodo `ListTopicsCommand` della classe client `SNS`. Per chiamare il `ListTopicsCommand` metodo, crea una funzione asincrona che richiama un oggetto di servizio Amazon SNS, passando l'oggetto parameters. Il file `data` restituito contiene una matrice del tuo argomento Amazon Resource Names (ARNs).

```
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;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui. GitHub

Eliminazione di un argomento

In questo esempio, usa un modulo Node.js per eliminare un argomento di Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome `snsClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `delete-topic.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto contenente il parametro `TopicArn` dell'argomento da eliminare per passare al metodo `DeleteTopicCommand` della classe client `SNS`. Per chiamare il `DeleteTopicCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> Sostituisci *`TOPIC_ARN`* con l'Amazon Resource Name (ARN) dell'argomento che stai eliminando.

```
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
  //   }
  // }
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Recupero degli attributi di argomento

In questo esempio, usa un modulo Node.js per recuperare gli attributi di un argomento Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `snsClient.js` Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `get-topic-attributes.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `TopicArn` di un argomento da eliminare per passare al metodo `GetTopicAttributesCommand` della classe client SNS. Per chiamare il `GetTopicAttributesCommand` metodo, è necessario richiamare un oggetto del servizio client Amazon SNS, passare l'oggetto parameters.

> **Note**
>
> Sostituisci *`TOPIC_ARN con`* l'ARN dell'argomento.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
```

```
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Impostazione degli attributi di argomento

In questo esempio, usa un modulo Node.js per impostare gli attributi mutabili di un argomento Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `snsClient.js` Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `set-topic-attributes.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente i parametri per l'aggiornamento dell'attributo, inclusi il parametro `TopicArn` dell'argomento di cui desideri impostare gli attributi, il nome dell'attributo da impostare e il nuovo valore per l'attributo. È possibile impostare solo gli attributi `Policy`, `DisplayName` e `DeliveryPolicy`. Trasferisci i parametri al metodo `SetTopicAttributesCommand` della classe client SNS. Per chiamare il `SetTopicAttributesCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> *Sostituisci ATTRIBUTE_NAME con il nome dell'attributo che stai impostando, TOPIC_ARN con l'Amazon Resource Name (ARN) dell'argomento di cui desideri impostare gli attributi e NEW_ATTRIBUTE_VALUE con il nuovo valore per quell'attributo.*

```
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;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Pubblicazione di messaggi in Amazon SNS

Questo esempio di codice di Node.js illustra:

- Come pubblicare messaggi su un argomento di Amazon SNS.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per pubblicare messaggi da Amazon SNS su endpoint, e-mail o numeri di telefono tematici. I moduli Node.js utilizzano l'SDK per JavaScript inviare messaggi utilizzando questo metodo della SNS classe client:

- `PublishCommand`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi. JavaScript Sintassi ES6/CommonJS

Pubblicazione di un messaggio in un argomento SNS

In questo esempio, usa un modulo Node.js per pubblicare un messaggio su un argomento di Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome `snsClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `publish-topic.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente i parametri per la pubblicazione di un messaggio, incluso il testo del messaggio e l'Amazon Resource Name (ARN) di Amazon SNSTopic. Per i dettagli sugli attributi SMS disponibili, consulta SetSMSAttributes.

Passa i parametri al `PublishCommand` metodo della classe SNS client. Crea una funzione asincrona richiamando un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> Sostituisci *`MESSAGE_TEXT`* con il testo del messaggio e *`TOPIC_ARN con l'ARN`* dell'argomento SNS.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *                                                 if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
```

```
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui su GitHub.

Gestione degli abbonamenti in Amazon SNS

Questo esempio di codice di Node.js illustra:

- Come elencare tutti gli abbonamenti a un argomento di Amazon SNS.
- Come sottoscrivere un indirizzo e-mail, un endpoint dell'applicazione o una AWS Lambda funzione a un argomento di Amazon SNS.
- Come annullare l'iscrizione agli argomenti di Amazon SNS.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per pubblicare messaggi di notifica su argomenti di Amazon SNS. I moduli Node.js utilizzano l'SDK per JavaScript gestire gli argomenti utilizzando questi metodi della classe SNS client:

- `ListSubscriptionsByTopicCommand`
- `SubscribeCommand`
- `ConfirmSubscriptionCommand`

- `UnsubscribeCommand`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi. JavaScript Sintassi ES6/CommonJS

Elenco di sottoscrizioni a un argomento

In questo esempio, usa un modulo Node.js per elencare tutte le sottoscrizioni a un argomento di Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `snsClient.js` Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `list-subscriptions-by-topic.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `TopicArn` per l'argomento di cui si desideri elencare le sottoscrizioni. Trasferisci i parametri al metodo `ListSubscriptionsByTopicCommand` della classe client SNS. Per chiamare il `ListSubscriptionsByTopicCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS e passa l'oggetto parameters.

> **Note**
>
> Sostituisci *`TOPIC_ARN`* con l'Amazon Resource Name (ARN) per l'argomento di cui desideri elencare gli abbonamenti.

```
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;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node list-subscriptions-by-topic.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Sottoscrizione di un indirizzo e-mail a un argomento

In questo esempio, usa un modulo Node.js per sottoscrivere un indirizzo e-mail in modo che riceva messaggi e-mail SMTP da un argomento di Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `snsClient.js` Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `subscribe-email.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il parametro `Protocol` per specificare il protocollo `email`, il parametro `TopicArn` per l'argomento a cui effettuare la sottoscrizione e un indirizzo e-mail come `Endpoint` del messaggio. Trasferisci i parametri al metodo `SubscribeCommand` della classe client `SNS`. Puoi utilizzare il `subscribe` metodo per sottoscrivere diversi endpoint a un argomento Amazon SNS, a seconda dei valori utilizzati per i parametri passati, come mostreranno altri esempi in questo argomento.

Per chiamare il `SubscribeCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS e passa l'oggetto parameters.

> **Note**
>
> Sostituisci *`TOPIC_ARN`* con l'Amazon Resource Name (ARN) per l'argomento e *`EMAIL_ADDRESS con l'indirizzo e-mail`* a cui iscriverti.

```
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'
  // }
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node subscribe-email.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Conferma delle sottoscrizioni

In questo esempio, utilizzate un modulo Node.js per verificare l'intenzione del proprietario di un endpoint di ricevere e-mail convalidando il token inviato all'endpoint con una precedente azione di sottoscrizione.

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `snsClient.js` Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

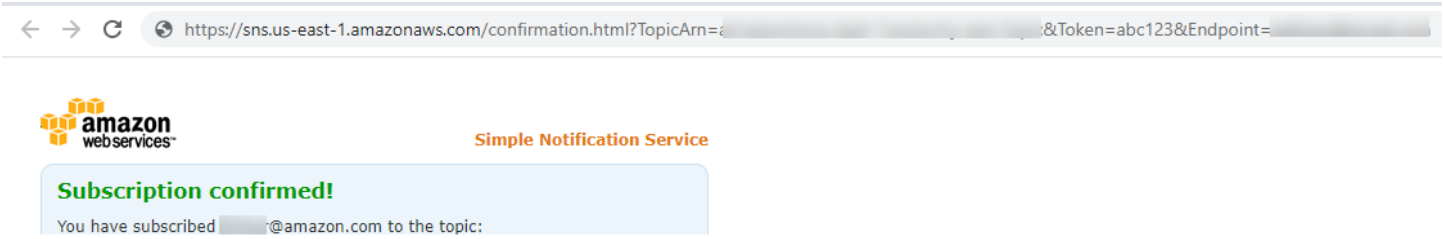
```
import { SNSClient } from "@aws-sdk/client-sns";

// 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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `confirm-subscription.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Definite i parametri, incluso `TOPIC_ARN` e`TOKEN`, e definite un valore di `TRUE` o `FALSE` per`AuthenticateOnUnsubscribe`.

Il token è un token di breve durata inviato al proprietario di un endpoint durante un'azione precedente`SUBSCRIBE`. Ad esempio, per un endpoint di posta elettronica, `TOKEN` si trova nell'URL dell'e-mail di conferma dell'iscrizione inviata al proprietario dell'e-mail. Ad esempio, `abc123` è il token nel seguente URL.



Per chiamare il `ConfirmSubscriptionCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> Sostituisci *`TOPIC_ARN`* con l'Amazon Resource Name (ARN) per l'argomento, *`TOKEN`* con il valore del token dell'URL inviato al proprietario dell'endpoint in un'`Subscribe`azione precedente e definisci *`AuthenticateOnUnsubscribe`*. con un valore di o. `TRUE` `FALSE`

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node confirm-subscription.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Sottoscrizione di un endpoint di applicazione a un argomento

In questo esempio, usa un modulo Node.js per sottoscrivere un endpoint di applicazione mobile in modo che riceva notifiche da un argomento di Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `snsClient.js` Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `subscribe-app.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei moduli e dei pacchetti richiesti.

Crea un oggetto contenente il `Protocol` parametro `TopicArn` per specificare il `application` protocollo, l'argomento a cui sottoscrivere e l'Amazon Resource Name (ARN) di un endpoint di applicazione mobile per il parametro. `Endpoint` Trasferisci i parametri al metodo `SubscribeCommand` della classe client SNS.

Per chiamare il `SubscribeCommand` metodo, crea una funzione asincrona che richiama un oggetto di servizio Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> Sostituisci *TOPIC_ARN* con l'Amazon Resource Name (ARN) per l'argomento e *MOBILE_ENDPOINT_ARN con l'endpoint* a cui stai sottoscrivendo l'argomento.

```
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;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node subscribe-app.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Sottoscrizione di una funzione Lambda a un argomento

In questo esempio, usa un modulo Node.js per sottoscrivere una AWS Lambda funzione in modo che riceva notifiche da un argomento di Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome `snsClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `subscribe-lambda.js`. Configura l'SDK come mostrato in precedenza.

Crea un oggetto contenente il `Protocol` parametro, specificando il `lambda` protocollo, l'`TopicArn`argomento a cui sottoscrivere e l'Amazon Resource Name (ARN) di AWS Lambda una funzione come `Endpoint` parametro. Trasferisci i parametri al metodo `SubscribeCommand` della classe client SNS.

Per chiamare il `SubscribeCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> Sostituisci *`TOPIC_ARN`* con Amazon Resource Name (ARN) per l'argomento e *`LAMBDA_FUNCTION_ARN con l'Amazon Resource Name (ARN) della funzione 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;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node subscribe-lambda.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Annullamento della sottoscrizione a un argomento

In questo esempio, usa un modulo Node.js per annullare l'iscrizione a un argomento Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome del file. `snsClient.js` Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `unsubscribe.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti.

Crea un oggetto contenente il `SubscriptionArn` parametro, specificando l'Amazon Resource Name (ARN) dell'abbonamento per annullare l'iscrizione. Trasferisci i parametri al metodo `UnsubscribeCommand` della classe client SNS.

Per chiamare il `UnsubscribeCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> Sostituisci *`TOPIC_SUBSCRIPTION_ARN con`* l'Amazon Resource Name (ARN) dell'abbonamento per annullare l'iscrizione.

```
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;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node unsubscribe.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Invio di messaggi SMS con Amazon SNS

Questo esempio di codice di Node.js illustra:

- Come ottenere e impostare le preferenze di messaggistica SMS per Amazon SNS.
- Come controllare un numero di telefono per verificare se è stata disattivata la ricezione di messaggi SMS.
- Come ottenere un elenco di numeri di telefono per cui è stata disattivata la ricezione di messaggi SMS.
- Come inviare un messaggio SMS.

Lo scenario

Puoi utilizzare Amazon SNS; per inviare messaggi SMS a dispositivi abilitati. Puoi inviare un messaggio direttamente a un numero di telefono oppure inviarlo a più numeri contemporaneamente sottoscrivendo quei numeri a un argomento e inviando il messaggio all'argomento.

In questo esempio, utilizzi una serie di moduli Node.js per pubblicare messaggi di testo SMS da Amazon SNS a dispositivi abilitati agli SMS. I moduli Node.js utilizzano l'SDK per JavaScript pubblicare messaggi SMS utilizzando questi metodi della classe client: SNS

- `GetSMSAttributesCommand`
- `SetSMSAttributesCommand`
- `CheckIfPhoneNumberIsOptedOutCommand`
- `ListPhoneNumbersOptedOutCommand`
- `PublishCommand`

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi. JavaScript Sintassi ES6/CommonJS

Recupero degli attributi SMS

Usa Amazon SNS per specificare le preferenze per la messaggistica SMS, ad esempio il modo in cui le consegne sono ottimizzate (in termini di costi o per una consegna affidabile), il limite di spesa mensile, il modo in cui vengono registrate le consegne dei messaggi e se abbonarsi ai report giornalieri sull'utilizzo degli SMS. Queste preferenze vengono recuperate e impostate come attributi SMS per Amazon SNS.

In questo esempio, usa un modulo Node.js per ottenere gli attributi SMS correnti in Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome `snsClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `get-sms-attributes.js`.

Configura l'SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Crea un oggetto contenente i parametri per ottenere attributi SMS, inclusi i nomi dei singoli attributi da recuperare. Per i dettagli sugli attributi SMS disponibili, consulta setSMSAttributes nel riferimento all'API di Amazon Simple Notification Service.

Questo esempio recupera l'attributo `DefaultSMSType`, che controlla se i messaggi SMS vengono inviati come `Promotional` per ottimizzare il recapito dei messaggi e permettere di contenere i costi, oppure come `Transactional` per ottimizzare il recapito dei messaggi e ottenere la massima affidabilità. Trasferisci i parametri al metodo `SetTopicAttributesCommand` della classe client `SNS`. Per chiamare il `SetSMSAttributesCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> Sostituisci *`ATTRIBUTE_NAME con il nome dell'attributo`*.

```
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;
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node get-sms-attributes.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Impostazione degli attributi SMS

In questo esempio, usa un modulo Node.js per ottenere gli attributi SMS correnti in Amazon SNS.

Crea una `libs` directory e crea un modulo Node.js con il nome `snsClient.js` del file. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `set-sms-attribute-type.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Crea un oggetto contenente i parametri per impostare gli attributi SMS, inclusi i nomi dei singoli attributi da impostare e i valori da impostare per ciascuno di essi. Per i dettagli sugli attributi SMS disponibili, consulta setSMSAttributes nel riferimento all'API di Amazon Simple Notification Service.

Questo esempio imposta l'attributo `DefaultSMSType` su `Transactional`, ottimizzando il recapito dei messaggi per ottenere la massima affidabilità. Trasferisci i parametri al metodo `SetTopicAttributesCommand` della classe client SNS. Per chiamare il `SetSMSAttributesCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

```
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;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node set-sms-attribute-type.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Controllare se un numero di telefono è stato disattivato

In questo esempio, utilizza un modulo Node.js per controllare un numero di telefono per verificare se è stata disattivata la ricezione di messaggi SMS.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`snsClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `check-if-phone-number-is-opted-out.js`. Configura l'SDK come mostrato in precedenza. Crea un oggetto contenente il numero di telefono da controllare come parametro.

Questo esempio imposta il parametro `PhoneNumber` per specificare il numero di telefono da verificare. Trasferisci l'oggetto al metodo `CheckIfPhoneNumberIsOptedOutCommand` della classe client `SNS`: Per chiamare il `CheckIfPhoneNumberIsOptedOutCommand` metodo, crea una

funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> 1.
>
> Sostituisci *PHONE_NUMBER con il numero di telefono*.

```
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;
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node check-if-phone-number-is-opted-out.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Elenco di numeri di telefono disattivati

In questo esempio, utilizza un modulo Node.js per ottenere un elenco di numeri di telefono per cui è stata disattivata la ricezione di messaggi SMS.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`snsClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `list-phone-numbers-opted-out.js`. Configura l'SDK come mostrato in precedenza. Crea un oggetto vuoto come parametro.

Trasferisci l'oggetto al metodo `ListPhoneNumbersOptedOutCommand` della classe client SNS: Per chiamare il `ListPhoneNumbersOptedOutCommand` metodo, crea una funzione asincrona che richiama un oggetto del servizio client Amazon SNS, passando l'oggetto parameters.

```
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;
};
```

Per eseguire l'esempio, inserisci quanto segue al prompt dei comandi.

```
node list-phone-numbers-opted-out.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Pubblicazione di un messaggio SMS

In questo esempio, utilizza un modulo Node.js per inviare un messaggio SMS a un numero di telefono.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`snsClient.js`. Copia e incolla il codice seguente, che crea l'oggetto client Amazon SNS. Sostituisci *`REGION`* con la tua AWS regione.

```
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({});
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `publish-sms.js`. Configura l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Crea un oggetto contenente i parametri `Message` e `PhoneNumber`.

Quando invii un SMS, ricorda di specificare il numero di telefono utilizzando il formato E.164. E.164 è uno standard per la struttura del numero di telefono utilizzato per le telecomunicazioni internazionali. I numeri di telefono che seguono questo formato possono avere un massimo di 15 cifre e sono preceduti dal segno più (+) e dal prefisso internazionale. Ad esempio, un numero di telefono negli Stati Uniti in formato E.164 verrà visualizzato come + 1001XXX5550100.

Questo esempio imposta il parametro `PhoneNumber` per specificare il numero di telefono a cui inviare il messaggio. Trasferisci l'oggetto al metodo `PublishCommand` della classe client SNS: Per chiamare il `PublishCommand` metodo, crea una funzione asincrona che richiama un oggetto di servizio Amazon SNS, passando l'oggetto parameters.

> **Note**
>
> *Sostituisci TEXT_MESSAGE con il messaggio di testo e PHONE_NUMBER con il numero di telefono.*

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *                                                 if you are using the `json`
 `MessageStructure`.
 * @param {*} phoneNumber - The phone number to send the message to.
 */
export const publish = async (
  message = "Hello from SNS!",
  phoneNumber = "+15555555555",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      // One of PhoneNumber, TopicArn, or TargetArn must be specified.
      PhoneNumber: phoneNumber,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '7410094f-efc7-5f52-af03-54737569ab77',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node publish-sms.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Esempi di Amazon Transcribe

Amazon Transcribe consente agli sviluppatori di aggiungere facilmente funzionalità di sintesi vocale alle proprie applicazioni.

L' JavaScript API per Amazon Transcribe è esposta tramite TranscribeServicela classe client.

Argomenti

- Esempi di Amazon Transcribe
- Amazon Transcribe: esempi medici

Esempi di Amazon Transcribe

In questo esempio, una serie di moduli Node.js vengono utilizzati per creare, elencare ed eliminare lavori di trascrizione utilizzando i seguenti metodi della classe client: `TranscribeService`

- `StartTranscriptionJobCommand`
- `ListTranscriptionJobsCommand`

- `DeleteTranscriptionJobCommand`

Per ulteriori informazioni sugli utenti di Amazon Transcribe, consulta la guida per sviluppatori di Amazon Transcribe.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi JavaScript Sintassi ES6/CommonJS

Avvio di un lavoro in Amazon Transcribe

Questo esempio dimostra come avviare un processo di trascrizione di Amazon Transcribe utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni, consulta. StartTranscriptionJobCommand

Create una `libs` directory e create un modulo Node.js con il nome del file`transcribeClient.js`. Copia e incolla il codice seguente al suo interno, per creare l'oggetto client Amazon Transcribe. Sostituisci *`REGION`* con la tua AWS regione.

```
const { TranscribeClient } = require("@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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `transcribe-create-job.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Create un oggetto parametrico, specificando i parametri richiesti. Avviate il lavoro utilizzando il `StartMedicalTranscriptionJobCommand` comando.

> **Note**
>
> Sostituisci *`MEDICAL_JOB_NAME con un nome`* per il lavoro di trascrizione. Per *`OUTPUT_BUCKET_NAME, specifica il bucket`* Amazon S3 in cui viene salvato l'output. Per *`JOB_TYPE`*, specifica i tipi di lavoro. Per *`SOURCE_LOCATION, specificate la posizione`* del file sorgente. Per *`SOURCE_FILE_LOCATION, specificate la posizione`* del file multimediale di input.

```
// 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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node transcribe-create-job.js
```

Questo codice di esempio può essere trovato qui. GitHub

Elenca le offerte di lavoro in Amazon Transcribe

Questo esempio mostra come elencare i lavori di trascrizione di Amazon Transcribe utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni su quali altre impostazioni puoi modificare, consulta. `ListTranscriptionJobCommand`

Create una `libs` directory e create un modulo Node.js con il nome del file`transcribeClient.js`. Copia e incolla il codice seguente al suo interno, per creare l'oggetto client Amazon Transcribe. Sostituisci *`REGION`* con la tua AWS regione.

```
const { TranscribeClient } = require("@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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `transcribe-list-jobs.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Create un oggetto parametrico con i parametri richiesti.

> **Note**
>
> Sostituisci *`KEY_WORD`* con una parola chiave che deve contenere il nome del job restituito.

```
// 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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node transcribe-list-jobs.js
```

Questo codice di esempio può essere trovato qui. GitHub

Eliminazione di un lavoro Amazon Transcribe

Questo esempio mostra come eliminare un processo di trascrizione di Amazon Transcribe utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni su optional, consulta. `DeleteTranscriptionJobCommand`

Create una `libs` directory e create un modulo Node.js con il nome del file`transcribeClient.js`. Copia e incolla il codice seguente al suo interno, per creare l'oggetto client Amazon Transcribe. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `transcribe-delete-job.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Specificate la AWS regione e il nome del lavoro che desiderate eliminare.

> **Note**
>
> Sostituite *JOB_NAME* con il nome del lavoro da eliminare.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

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

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node transcribe-delete-job.js
```

Questo codice di esempio può essere trovato qui. GitHub

Amazon Transcribe: esempi medici

In questo esempio, una serie di moduli Node.js vengono utilizzati per creare, elencare ed eliminare lavori di trascrizione medica utilizzando i seguenti metodi della classe client: `TranscribeService`

- `StartMedicalTranscriptionJobCommand`
- `ListMedicalTranscriptionJobsCommand`
- `DeleteMedicalTranscriptionJobCommand`

Per ulteriori informazioni sugli utenti di Amazon Transcribe, consulta la guida per sviluppatori di Amazon Transcribe.

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questi esempi dimostrano come importare/esportare oggetti e comandi del servizio client utilizzando ECMAScript6 (ES6).
>
> - Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> - Se preferisci usare la sintassi CommonJS, vedi JavaScript Sintassi ES6/CommonJS

Avvio di un lavoro di trascrizione medica con Amazon Transcribe

Questo esempio dimostra come avviare un processo di trascrizione medica di Amazon Transcribe utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni, vedere startMedicalTranscriptionJob.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`transcribeClient.js`. Copia e incolla il codice seguente al suo interno, per creare l'oggetto client Amazon Transcribe. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `transcribe-create-medical-job.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Create un oggetto parametrico, specificando i parametri richiesti. Inizia il lavoro medico usando il `StartMedicalTranscriptionJobCommand` comando.

> **Note**
>
> Sostituisci *`MEDICAL_JOB_NAME con un nome`* per il lavoro di trascrizione medica. Per *`OUTPUT_BUCKET_NAME, specifica il bucket`* Amazon S3 in cui viene salvato l'output. Per *`JOB_TYPE`*, specifica i tipi di lavoro. Per *`SOURCE_LOCATION, specificate la posizione`* del file sorgente. Per *`SOURCE_FILE_LOCATION, specificate la posizione`* del file multimediale di input.

```
// 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();
```

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node transcribe-create-medical-job.js
```

Questo codice di esempio può essere trovato qui. GitHub

Pubblicazione di offerte di lavoro nel settore medico in Amazon Transcribe

Questo esempio mostra come elencare i lavori di trascrizione di Amazon Transcribe utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni, consulta. ListTranscriptionMedicalJobsCommand

Create una `libs` directory e create un modulo Node.js con il nome del file`transcribeClient.js`. Copia e incolla il codice seguente al suo interno, per creare l'oggetto client Amazon Transcribe. Sostituisci *`REGION`* con la tua AWS regione.

```
const { TranscribeClient } = require("@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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `transcribe-list-medical-jobs.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Create un oggetto parametrico con i parametri richiesti ed elencate i lavori medici utilizzando il `ListMedicalTranscriptionJobsCommand` comando.

> **Note**
>
> Sostituisci *KEYWORD* con una parola chiave che deve contenere il nome dei lavori restituiti.

```
// 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();
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi.

```
node transcribe-list-medical-jobs.js
```

Questo codice di esempio può essere trovato qui. GitHub

Eliminazione di un lavoro medico in Amazon Transcribe

Questo esempio mostra come eliminare un processo di trascrizione di Amazon Transcribe utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni su optional, consulta. `DeleteTranscriptionMedicalJobCommand`

Create una `libs` directory e create un modulo Node.js con il nome del file`transcribeClient.js`. Copia e incolla il codice seguente al suo interno, per creare l'oggetto client Amazon Transcribe. Sostituisci *`REGION`* con la tua AWS regione.

```
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 };
```

Questo codice di esempio può essere trovato qui GitHub.

Crea un modulo Node.js con il nome del file `transcribe-delete-job.js`. Assicurati di configurare l'SDK come mostrato in precedenza, inclusa l'installazione dei client e dei pacchetti richiesti. Create un oggetto parametrico con i parametri richiesti ed eliminate il lavoro medico utilizzando il `DeleteMedicalJobCommand` comando.

> **Note**
>
> Sostituisci *`JOB_NAME`* con il nome del lavoro da eliminare.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
```

```
};

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

Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

```
node transcribe-delete-medical-job.js
```

Questo codice di esempio può essere trovato qui. GitHub

Configurazione di Node.js su un'istanza Amazon EC2

Uno scenario comune per l'utilizzo di Node.js con l'SDK per JavaScript consiste nel configurare ed eseguire un'applicazione Web Node.js su un'istanza Amazon Elastic Compute Cloud (Amazon EC2). In questo tutorial, creerai un'istanza Linux, ti connetterai a essa tramite SSH, quindi installerai Node.js per l'esecuzione su tale istanza.

Prerequisiti

Questo tutorial presuppone che tu abbia già avviato un'istanza Linux con un nome DNS pubblico raggiungibile da Internet e a cui puoi connetterti tramite SSH. Per ulteriori informazioni, consulta la Fase 1: Avvio di un'istanza nella Guida per l'utente di Amazon EC2 per le istanze Linux.

> **Important**
>
> Usa Amazon Linux 2023 Amazon Machine Image (AMI) per lanciare una nuova istanza Amazon EC2.

È inoltre necessario aver configurato il gruppo di sicurezza per consentire le connessioni SSH (porta 22), HTTP (porta 80) e HTTPS (porta 443). Per ulteriori informazioni su questi prerequisiti, consulta Configurazione con Amazon EC2 nella Amazon EC2 User Guide for Linux Instances.

Procedura

La procedura seguente ti consente di installare Node.js su un'istanza Amazon Linux. Puoi utilizzare questo server per l'hosting di un'applicazione Web Node.js.

Per configurare Node.js sulla tua istanza Linux

1. Connettersi all'istanza Linux come `ec2-user` tramite SSH.
2. Installa node version manager (`nvm`) digitando quanto segue nella riga di comando.

   > **Warning**
   >
   > AWS non controlla il codice seguente. Prima di eseguirlo, assicurati di verificarne l'autenticità e l'integrità. Ulteriori informazioni su questo codice sono disponibili nel repository nvm. GitHub

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
   ```

   Useremo `nvm` per installare Node.js perché `nvm` può installare più versioni di Node.js e permetterti di passare da una all'altra.
3. Carica `nvm` digitando quanto segue nella riga di comando.

   ```
   source ~/.bashrc
   ```
4. Usa nvm per installare l'ultima versione LTS di Node.js digitando quanto segue nella riga di comando.

   ```
   nvm install --lts
   ```

   L'installazione di Node.js installa anche il Node Package Manager (`npm`) in modo da poter installare moduli aggiuntivi secondo necessità.
5. Verificare che Node.js sia installato e correttamente in esecuzione digitando quanto segue nella riga di comando.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Viene visualizzato il seguente messaggio che mostra la versione di Node.js in esecuzione.

```
Running Node.js VERSION
```

> **Note**
>
> L'installazione del nodo si applica solo alla sessione corrente di Amazon EC2. Se si riavvia la sessione CLI, è necessario utilizzare nuovamente nvm per abilitare la versione del nodo installata. Se l'istanza viene terminata, devi installare nuovamente il nodo. L'alternativa è creare un'Amazon Machine Image (AMI) dell'istanza Amazon EC2 una volta ottenuta la configurazione che desideri mantenere, come descritto nel seguente argomento.

Creazione di un'Amazon Machine Image (AMI)

Dopo aver installato Node.js su un'istanza Amazon EC2, puoi creare un'Amazon Machine Image (AMI) da quell'istanza. La creazione di un'AMI semplifica il provisioning di più istanze Amazon EC2 con la stessa installazione di Node.js. Per ulteriori informazioni sulla creazione di un'AMI da un'istanza esistente, consulta Creazione di un'AMI Linux supportata da Amazon EBS nella Guida per l'utente di Amazon EC2 per istanze Linux.

Risorse correlate

Per ulteriori informazioni sui comandi e sul software utilizzati in questo argomento, consulta le seguenti pagine Web:

- Node version manager (`nvm`): vedere nvm repo on. GitHub
- Node Package Manager (`npm`) —Vedi il sito web di npm.

Crea un'app per inviare dati a DynamoDB

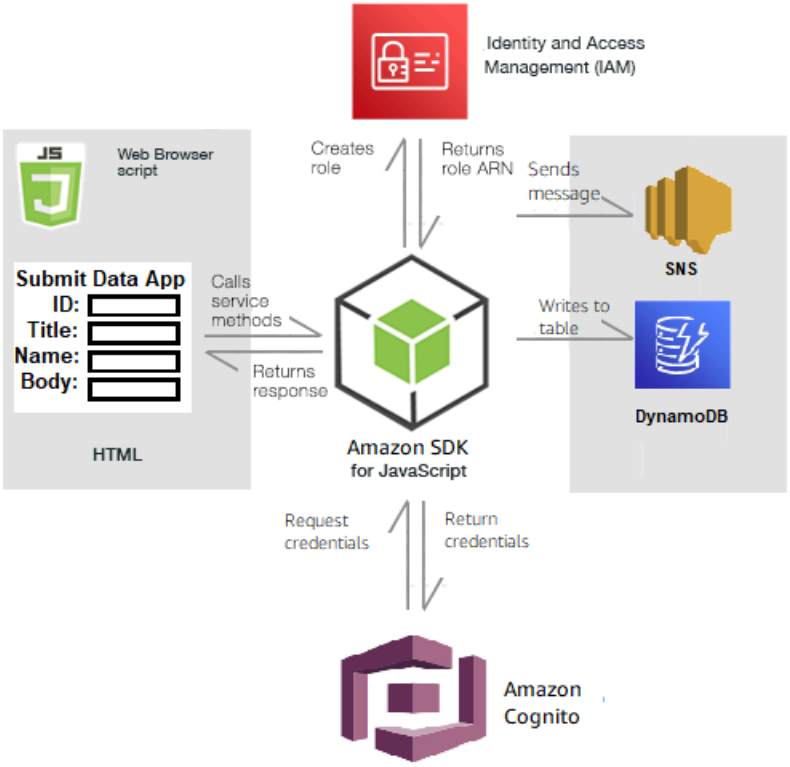
Questo tutorial interservizio di Node.js mostra come creare un'app che consenta agli utenti di inviare dati a una tabella Amazon DynamoDB. Questa app utilizza i seguenti servizi:

- AWS Identity and Access Management(IAM) e Amazon Cognito per autorizzazioni e autorizzazioni.

- Amazon DynamoDB (DynamoDB) per creare e aggiornare le tabelle.
- Amazon Simple Notification Service (Amazon SNS) per notificare all'amministratore dell'app quando un utente aggiorna la tabella.

Lo scenario

In questo tutorial, una pagina HTML fornisce un'applicazione basata su browser per l'invio di dati a una tabella Amazon DynamoDB. L'app utilizza Amazon SNS per notificare all'amministratore dell'app quando un utente aggiorna la tabella.



Per creare l'app:

1. Prerequisiti
2. Effettuare il provisioning delle risorse
3. Crea il codice HTML
4. Crea lo script del browser
5. Fasi successive

Prerequisiti

Completate le seguenti attività preliminari:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

Crea le risorse AWS

Questa app richiede le seguenti risorse:

- AWS Identity and Access Management(IAM) Ruolo utente Amazon Cognito non autenticato con le seguenti autorizzazioni:
  - sns:Publish
  - dynamodb: PutItem
- Una tabella DynamoDB.

Puoi creare queste risorse manualmente nella AWS console, ma ti consigliamo di effettuare il provisioning di queste risorse utilizzando AWS CloudFormation quanto descritto in questo tutorial.

Crea le AWS risorse utilizzando AWS CloudFormation

AWS CloudFormation consente di creare ed effettuare il provisioning delle distribuzioni dell'infrastruttura AWS in modo ripetuto e prevedibile. Per ulteriori informazioni su AWS CloudFormation, consulta la AWS CloudFormationGuida per l'utente di .

Per creare lo AWS CloudFormation stack usando: AWS CLI

1. Installa e configura le AWS CLI seguenti istruzioni contenute nella Guida per l'AWS CLIutente.
2. Create un file denominato `setup.yaml` nella directory principale della cartella del progetto e copiatene il contenuto. GitHub

> **Note**
>
> Il AWS CloudFormation modello è stato generato utilizzando quello AWS CDK disponibile qui GitHub. Per ulteriori informazioni su AWS CDK, consulta la Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK).

3. Esegui il seguente comando dalla riga di comando, sostituendo *STACK_NAME con un nome* univoco per lo stack e *REGION nella tua regione*. AWS

   > **Important**
   >
   > Il nome dello stack deve essere univoco all'interno di una regione e di un account. AWS AWS È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

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

   Per ulteriori informazioni sui parametri dei `create-stack` comandi, consultate la guida di riferimento ai AWS CLI comandi e la Guida per l'AWS CloudFormationutente.

   Per visualizzare le risorse create, apri AWS CloudFormation nella console di AWS gestione, scegli lo stack e seleziona la scheda Risorse.

4. Quando lo stack viene creato, usa il AWS SDK for JavaScript per popolare la tabella DynamoDB, come descritto in. Compilazione della tabella

Compilazione della tabella

Per compilare la tabella, create innanzitutto una directory denominata`libs`, in essa create un file denominato `dynamoClient.js` e incollate il contenuto sottostante. Sostituisci *REGION* con la tua AWS regione e sostituisci *IDENTITY_POOL_ID con un* ID Amazon Cognito Identity Pool. Questo crea l'oggetto client 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 DynaomDB service client object.
const dynamoClient = new DynamoDBClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { dynamoClient };
```

Questo codice è disponibile qui. GitHub

Quindi, crea una `dynamoAppHelperFiles` cartella nella cartella del tuo progetto, crea un file `update-table.js` al suo interno e copia il contenuto qui GitHub dentro.

```
// 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();
```

Esegui il seguente comando dalla riga di comando.

```
node update-table.js
```

Questo codice è disponibile qui GitHub.

Crea una pagina front-end per l'app

Qui puoi creare la pagina del browser HTML front-end per l'app.

Crea una `DynamoDBApp` directory, crea un file denominato `index.html` e copia il codice da qui in poi. GitHub L'`script`elemento aggiunge il `main.js` file, che contiene tutto il necessario JavaScript per l'esempio. Il `main.js` file verrà creato più avanti in questo tutorial. Il codice rimanente `index.html` crea la pagina del browser che acquisisce i dati inseriti dagli utenti.

Questo codice di esempio può essere trovato qui su GitHub.

Crea lo script del browser

Innanzitutto, create gli oggetti client di servizio richiesti per l'esempio. Create una `libs` directory`snsClient.js`, create e incollate il codice seguente al suo interno. Sostituisci *`REGION`* e *`IDENTITY_POOL_ID`* in ciascuno.

> **Note**
>
> Usa l'ID del pool di identità di Amazon Cognito in cui hai creato. Crea le risorse 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 };
```

Questo codice è disponibile qui. GitHub .

Per creare lo script del browser per questo esempio, in una cartella denominataDynamoDBApp, create un modulo Node.js con il nome del file `add_data.js` e incollate il codice seguente al suo interno. La `submitData` funzione invia i dati a una tabella DynamoDB e invia un SMS all'amministratore dell'app tramite Amazon SNS.

Nella `submitData` funzione, dichiara le variabili per il numero di telefono di destinazione, i valori immessi nell'interfaccia dell'app e per il nome del bucket Amazon S3. Quindi, crea un oggetto parametrico per aggiungere un elemento alla tabella. Se nessuno dei valori è vuoto, `submitData` aggiunge l'elemento alla tabella e invia il messaggio. Ricordati di rendere disponibile la funzione al browser, 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;
```

Questo codice di esempio può essere trovato qui su GitHub.

Infine, esegui quanto segue al prompt dei comandi per raggruppare il file JavaScript per questo esempio in un file denominato: `main.js`

```
webpack add_data.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> Per informazioni sull'installazione di webpack, consulta. Raggruppa le applicazioni con webpack

Per eseguire l'app, apri `index.html` il browser.

Elimina le risorse

Come indicato all'inizio di questo tutorial, assicurati di terminare tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti vengano addebitati costi. Puoi farlo eliminando lo AWS CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

1. Apri il file AWS CloudFormationnella console di AWS gestione.
2. Apri la pagina Stacks e seleziona lo stack.
3. Scegli Elimina.

Per altri esempi AWS inter-service, consulta AWS SDK for JavaScript Esempi interservice.

Crea un'app di trascrizione con utenti autenticati

In questo tutorial, apprenderai come:

- Implementa l'autenticazione utilizzando un pool di identità Amazon Cognito per accettare utenti federati con un pool di utenti Amazon Cognito.
- Usa Amazon Transcribe per trascrivere e visualizzare le registrazioni vocali nel browser.

Lo scenario

L'app consente agli utenti di registrarsi con un'e-mail e un nome utente univoci. Dopo la conferma della loro e-mail, possono registrare messaggi vocali che vengono automaticamente trascritti e visualizzati nell'app.

Come funziona

L'app utilizza due bucket Amazon S3, uno per ospitare il codice dell'applicazione e l'altro per archiviare le trascrizioni. L'app utilizza un pool di utenti Amazon Cognito per autenticare gli utenti. Gli utenti autenticati dispongono delle autorizzazioni IAM per accedere ai servizi richiesti. AWS

La prima volta che un utente registra un messaggio vocale, Amazon S3 crea una cartella univoca con il nome dell'utente nel bucket Amazon S3 per l'archiviazione delle trascrizioni. Amazon Transcribe trascrive il messaggio vocale in testo e lo salva in JSON nella cartella dell'utente. Quando l'utente aggiorna l'app, le relative trascrizioni vengono visualizzate e sono disponibili per il download o l'eliminazione.

Il completamento di questo tutorial richiede circa 30 minuti.

Fasi

Per creare l'app:

1. Prerequisiti
2. Crea le risorse AWS
3. Crea il codice HTML
4. Preparare lo script del browser
5. Esegui l'app
6. Elimina le risorse

Prerequisiti

- Configura l'ambiente di progetto per eseguire questi JavaScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questo esempio utilizza ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Crea le risorse AWS

Questa sezione descrive come fornire AWS risorse per questa app utilizzandoAWS Cloud Development Kit (AWS CDK).

> **Note**
>
> AWS CDKÈ un framework di sviluppo software che consente di definire le risorse delle applicazioni cloud. Per ulteriori informazioni, consulta la Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK).

Per creare risorse per l'app, utilizza il modello qui sotto GitHub per creare uno AWS CDK stack utilizzando la console di gestione dei servizi AWS Web o il AWS CLI. Per istruzioni su come modificare lo stack o su come eliminare lo stack e le risorse associate una volta terminato il tutorial, consulta questa pagina. GitHub

> **Note**
>
> Il nome dello stack deve essere univoco all'interno di una AWS regione e di un account. AWS È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

Lo stack risultante fornisce automaticamente le seguenti risorse.

- Un pool di identità Amazon Cognito con un ruolo utente autenticato.
- Una policy IAM con autorizzazioni per Amazon S3 e Amazon Transcribe è allegata al ruolo utente autenticato.
- Un pool di utenti Amazon Cognito che consente agli utenti di registrarsi e accedere all'app.
- Un bucket Amazon S3 per ospitare i file dell'applicazione.
- Un bucket Amazon S3 per archiviare le trascrizioni.

> **Important**
>
> Questo bucket Amazon S3 consente l'accesso pubblico a READ (LIST), che consente a chiunque di elencare gli oggetti all'interno del bucket e di utilizzare potenzialmente in modo improprio le informazioni. Se non elimini questo bucket Amazon S3 subito dopo aver completato il tutorial, ti consigliamo vivamente di attenerti alle best practice di sicurezza di Amazon S3 nella Amazon Simple Storage Service User Guide.

Crea il codice HTML

Crea un `index.html` file e copia e incolla il contenuto seguente al suo interno. La pagina presenta un pannello di pulsanti per la registrazione dei messaggi vocali e una tabella che mostra i messaggi precedentemente trascritti dall'utente corrente. Il tag script alla fine dell'`body`elemento richiama il`main.js`, che contiene tutto lo script del browser per l'app. Il file main.js viene creato utilizzando Webpack, come descritto nella sezione seguente di questo 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>
```

Questo esempio di codice è disponibile qui. GitHub

Preparate lo script del browser

Esistono tre file,`index.html`, e `recorder.jshelper.js`, che è necessario raggruppare in un unico file `main.js` utilizzando Webpack. Questa sezione descrive in dettaglio solo le funzioni per cui viene utilizzato l'SDK JavaScript, disponibile qui. `index.js` GitHub

> **Note**
>
> `recorder.js` e `helper.js` sono obbligatori ma, poiché non contengono codice Node.js, sono spiegati rispettivamente nei commenti in linea qui e qui. GitHub

Definite innanzitutto i parametri. `COGNITO_ID` è l'endpoint per il pool di utenti di Amazon Cognito che hai creato Crea le risorse AWS nell'argomento di questo tutorial. È formattato. `cognito-idp.`*`AWS_REGION`*`.amazonaws.com/`*`USER_POOL_ID`* L'id del pool di utenti è *`ID_TOKEN`* nel token AWS delle credenziali, che viene rimosso dall'URL dell'app dalla `getToken` funzione nel file 'helper.js'. Questo token viene passato alla `loginData` variabile, che fornisce gli accessi agli oggetti client Amazon Transcribe e Amazon S3. Sostituisci *«REGION»* con la AWS regione e *«BUCKET»* con *«IDENTITY_POOL_ID» con la pagina di esempio del IdentityPoolId pool* di identità Amazon Cognito che hai creato per questo esempio. Questo viene passato anche a ciascun oggetto client.

```
// Import the required AWS SDK clients and commands for Node.js
import "./helper.js";
import "./recorder.js";
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import {
  CognitoIdentityProviderClient,
  GetUserCommand,
} from "@aws-sdk/client-cognito-identity-provider";
import { S3RequestPresigner } from "@aws-sdk/s3-request-presigner";
import { createRequest } from "@aws-sdk/util-create-request";
import { formatUrl } from "@aws-sdk/util-format-url";
import {
  TranscribeClient,
  StartTranscriptionJobCommand,
} from "@aws-sdk/client-transcribe";
import {
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
  ListObjectsCommand,
  DeleteObjectCommand,
} from "@aws-sdk/client-s3";
import fetch from "node-fetch";
```

```
// Set the parameters.
// 'COGINTO_ID' has the format 'cognito-idp.eu-west-1.amazonaws.com/COGNITO_ID'.
let COGNITO_ID = "COGNITO_ID";
// Get the Amazon Cognito ID token for the user. 'getToken()' is in 'helper.js'.
let idToken = getToken();
let loginData = {
  [COGNITO_ID]: idToken,
};

const params = {
  Bucket: "BUCKET", // The Amazon Simple Storage Solution (S3) bucket to store the
 transcriptions.
  Region: "REGION", // The AWS Region
  identityPoolID: "IDENTITY_POOL_ID", // Amazon Cognito Identity Pool ID.
};

// Create an Amazon Transcribe service client object.
const client = new TranscribeClient({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});

// Create an Amazon S3 client object.
const s3Client = new S3Client({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});
```

Quando la pagina HTML viene caricata, `updateUserInterface` crea una cartella con il nome dell'utente nel bucket Amazon S3 se è la prima volta che l'utente accede all'app. In caso contrario, aggiorna l'interfaccia utente con tutte le trascrizioni delle sessioni precedenti dell'utente.

```
window.onload = async () => {
```

```
  // Set the parameters.
  const userParams = {
    // Get the access token. 'GetAccessToken()' is in 'helper.js'.
    AccessToken: getAccessToken(),
  };
  // Create a CognitoIdentityProviderClient client object.
  const client = new CognitoIdentityProviderClient({ region: params.Region });
  try {
    const data = await client.send(new GetUserCommand(userParams));
    const username = data.Username;
    // Export username for use in 'recorder.js'.
    exports.username = username;
    try {
      // If this is user's first sign-in, create a folder with user's name in Amazon S3
bucket.
      // Otherwise, no effect.
      const Key = `${username}/`;
      try {
        const data = await s3Client.send(
          new PutObjectCommand({ Key: Key, Bucket: params.Bucket })
        );
        console.log("Folder created for user ", data.Username);
      } catch (err) {
        console.log("Error", err);
      }
      try {
        // Get a list of the objects in the Amazon S3 bucket.
        const data = await s3Client.send(
          new ListObjectsCommand({ Bucket: params.Bucket, Prefix: username })
        );
        // Create a variable for the list of objects in the Amazon S3 bucket.
        const output = data.Contents;
        // Loop through the objects, populating a row on the user interface for each
object.
        for (var i = 0; i < output.length; i++) {
          var obj = output[i];
          const objectParams = {
            Bucket: params.Bucket,
            Key: obj.Key,
          };
          // Get the name of the object from the Amazon S3 bucket.
          const data = await s3Client.send(new GetObjectCommand(objectParams));
          // Extract the body contents, a readable stream, from the returned data.
          const result = data.Body;
```

```
          // Create a variable for the string version of the readable stream.
          let stringResult = "";
          // Use 'yieldUnit8Chunks' to convert the readable streams into JSON.
          for await (let chunk of yieldUint8Chunks(result)) {
            stringResult += String.fromCharCode.apply(null, chunk);
          }
          // The setTimeout function waits while readable stream is converted into
 JSON.
          setTimeout(function () {
            // Parse JSON into human readable transcript, which will be displayed on
 user interface (UI).
            const outputJSON =
              JSON.parse(stringResult).results.transcripts[0].transcript;
            // Create name for transcript, which will be displayed.
            const outputJSONTime = JSON.parse(stringResult)
              .jobName.split("/")[0]
              .replace("-job", "");
            i++;
            //
            // Display the details for the transcription on the UI.
            // 'displayTranscriptionDetails()' is in 'helper.js'.
            displayTranscriptionDetails(
              i,
              outputJSONTime,
              objectParams.Key,
              outputJSON
            );
          }, 1000);
        }
      } catch (err) {
        console.log("Error", err);
      }
    } catch (err) {
      console.log("Error creating presigned URL", err);
    }
  } catch (err) {
    console.log("Error", err);
  }
};

// Convert readable streams.
async function* yieldUint8Chunks(data) {
  const reader = data.getReader();
  try {
```

```
    while (true) {
      const { done, value } = await reader.read();
      if (done) return;
      yield value;
    }
  } finally {
    reader.releaseLock();
  }
}
```

Quando l'utente registra un messaggio vocale per le trascrizioni, `upload` carica le registrazioni nel bucket Amazon S3. Questa funzione viene richiamata dal file. `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 trascrizione dall'interfaccia utente ed `deleteRow` elimina una trascrizione esistente dal bucket Amazon S3. Entrambi vengono attivati dal pulsante Elimina sull'interfaccia utente.

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

Infine, esegui quanto segue al prompt dei comandi per raggruppare il file JavaScript per questo esempio in un file denominato: `main.js`

```
webpack index.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> Per informazioni sull'installazione di webpack, consulta. Raggruppa le applicazioni con webpack

Esegui l'app

Puoi quindi visualizzare l'app nella posizione seguente.

```
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 semplifica l'esecuzione dell'app fornendo un collegamento nella console di gestione dei servizi AWS Web. Accedi semplicemente all'impostazione del client App del tuo pool di utenti Amazon Cognito e seleziona l'interfaccia utente Launch Hosted. L'URL dell'app ha il seguente formato.

> **Important**
>
> L'interfaccia utente ospitata utilizza per impostazione predefinita un tipo di risposta di «codice». Tuttavia, questo tutorial è progettato per il tipo di risposta «token», quindi è necessario modificarlo.

Eliminare le AWS risorse

Al termine del tutorial, è necessario eliminare le risorse in modo da non incorrere in addebiti inutili. Poiché hai aggiunto contenuti a entrambi i bucket Amazon S3, devi eliminarli manualmente. È quindi possibile eliminare le risorse rimanenti utilizzando la console di gestione dei servizi AWS Web o il. AWS CLI Le istruzioni su come modificare lo stack o su come eliminare lo stack e le risorse associate una volta terminato il tutorial, sono disponibili qui. GitHub

Richiamare Lambda con API Gateway

Puoi richiamare una funzione Lambda utilizzando Amazon API Gateway, AWS un servizio per la creazione, la pubblicazione, la manutenzione, il monitoraggio e la protezione di REST, WebSocket

HTTP e API su larga scala. Gli sviluppatori di API possono creare API in grado di accedere ad AWS o ad altri servizi Web, nonché ai dati archiviati in AWS Cloud. In qualità di sviluppatore di API Gateway, puoi creare API da utilizzare nelle tue applicazioni client. Per ulteriori informazioni, consulta Cos'è Amazon API Gateway.

AWS Lambdaè un servizio di elaborazione che consente di eseguire codice senza fornire o gestire server. È possibile creare funzioni Lambda in diversi linguaggi di programmazione. Per ulteriori informazioni su AWS Lambda, consulta Che cos'è AWS Lambda.

In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi servizi AWS per eseguire un caso d'uso specifico. Ad esempio, supponiamo che un'organizzazione invii un messaggio di testo mobile ai propri dipendenti per congratularsi con loro per la data del primo anniversario, come illustrato in questa illustrazione.

Il completamento dell'esempio dovrebbe richiedere circa 20 minuti.

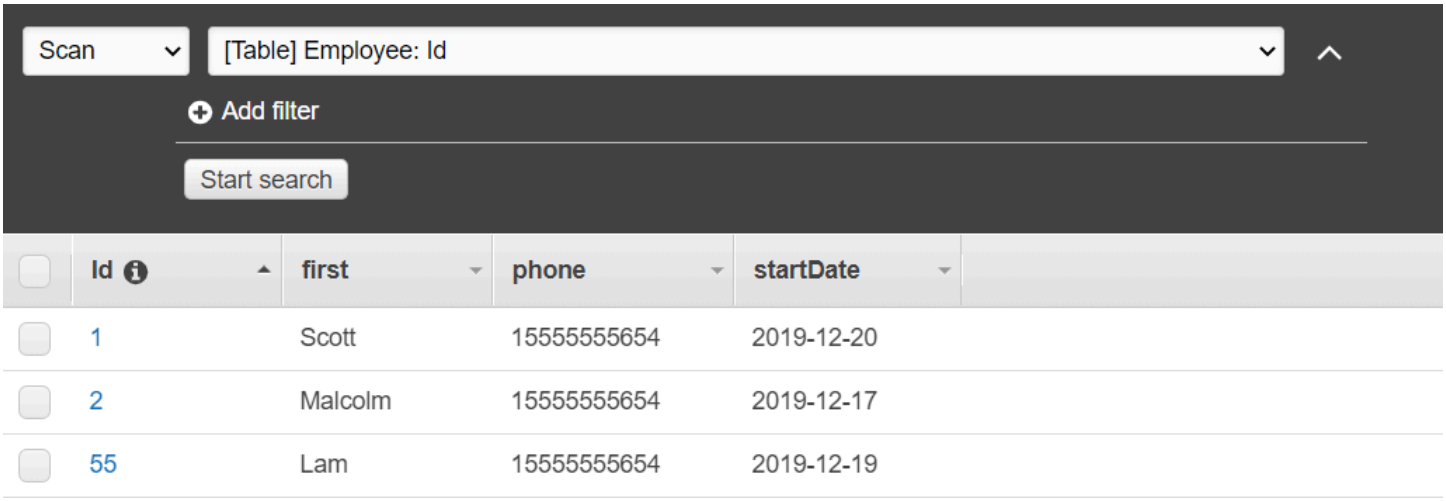
Questo esempio mostra come utilizzare la JavaScript logica per creare una soluzione che esegua questo caso d'uso. Ad esempio, imparerai a leggere un database per determinare quali dipendenti hanno raggiunto la data del primo anniversario, come elaborare i dati e inviare un messaggio di testo, il tutto utilizzando una funzione Lambda. Quindi imparerai come utilizzare API Gateway per richiamare questa AWS Lambda funzione utilizzando un endpoint Rest. Ad esempio, puoi richiamare la funzione Lambda utilizzando questo comando curl:

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/
employee"
```

Questo AWS tutorial utilizza una tabella Amazon DynamoDB denominata Employee che contiene questi campi.

- id: la chiave primaria per la tabella.
- firstName: nome del dipendente.

- telefono: numero di telefono del dipendente.
- StartDate: data di inizio del dipendente.



> **Important**
>
> Costo di completamento: i AWS servizi inclusi in questo documento sono inclusi nel piano AWS gratuito. Tuttavia, assicurati di terminare tutte le risorse dopo aver completato questo esempio per assicurarti che non ti venga addebitato alcun costo.

Per creare l'app:

1. Prerequisiti completi
2. Crea le risorse AWS
3. Preparare lo script del browser
4. Crea e carica la funzione Lambda
5. Implementa la funzione Lambda
6. Esegui l'app
7. Elimina le risorse

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

Crea le risorse AWS

Questo tutorial richiede le seguenti risorse:

- Una tabella Amazon DynamoDB `Employee` denominata con una chiave `Id` denominata e i campi mostrati nell'illustrazione precedente. Assicurati di inserire i dati corretti, incluso un telefono cellulare valido con cui desideri testare questo caso d'uso. Per ulteriori informazioni, consulta Creare una tabella.
- Un ruolo IAM con autorizzazioni allegate per eseguire funzioni Lambda.
- Un bucket Amazon S3 per ospitare la funzione Lambda.

Puoi creare queste risorse manualmente, ma ti consigliamo di effettuare il provisioning di queste risorse usando AWS CloudFormation come descritto in questo tutorial.

Crea le AWS risorse utilizzando AWS CloudFormation

AWS CloudFormation consente di creare ed effettuare il provisioning delle distribuzioni dell'infrastruttura AWS in modo ripetuto e prevedibile. Per ulteriori informazioni su AWS CloudFormation, consulta la AWS CloudFormationGuida per l'utente di .

Per creare lo AWS CloudFormation stack usando: AWS CLI

1. Installa e configura le AWS CLI seguenti istruzioni contenute nella Guida per l'AWS CLIutente.
2. Create un file denominato `setup.yaml` nella directory principale della cartella del progetto e copiatene il contenuto. GitHub

> **Note**
>
> Il AWS CloudFormation modello è stato generato utilizzando quello AWS CDK disponibile qui GitHub. Per ulteriori informazioni su AWS CDK, consulta la Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK).

3. Esegui il seguente comando dalla riga di comando, sostituendo *STACK_NAME con un nome* univoco per lo stack.

> **Important**
>
> Il nome dello stack deve essere univoco all'interno di una regione e di un account. AWS AWS È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

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

Per ulteriori informazioni sui parametri dei `create-stack` comandi, consultate la guida di riferimento ai AWS CLI comandi e la Guida per l'AWS CloudFormationutente.

4. Successivamente, compila la tabella seguendo la proceduraCompilazione della tabella.

Compilazione della tabella

Per compilare la tabella, create innanzitutto una directory denominata`libs`, in essa create un file denominato `dynamoClient.js` e incollate il contenuto sottostante.

```
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 };
```

Questo codice è disponibile qui su. GitHub

Quindi, crea un file denominato `populate-table.js` nella directory principale della cartella del tuo progetto e copia il contenuto qui GitHub dentro. Per uno degli elementi, sostituisci il valore della `phone` proprietà con un numero di cellulare valido nel formato E.164 e il valore della `startDate` con la data odierna.

Esegui il comando seguente dalla riga di comando.

```
node populate-table.js
```

```
const { BatchWriteItemCommand } = require ( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Set the parameters.
export const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};
```

```
export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Questo codice è disponibile qui GitHub.

Creazione della funzione AWS Lambda

Configurazione dell'SDK

Nella `libs` directory, crea file denominati `snsClient.js` and e `lambdaClient.js` incolla il contenuto seguente in questi file, rispettivamente.

```
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 };
```

Sostituisci *`REGION`* con AWS Region. Questo codice è disponibile qui 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 };
```

Sostituisci *`REGION`* con la AWS Region. Questo codice è disponibile qui GitHub.

Innanzitutto, importa i moduli e i comandi richiesti AWS SDK for JavaScript (v3). Quindi calcola la data odierna e assegnala a un parametro. Terzo, crea i parametri per. `ScanCommand` Sostituite *`TABLE_NAME`* con il nome della tabella creata nella Crea le risorse AWS sezione di questo esempio.

Il seguente frammento di codice mostra questa fase. (Per l'esempio completo, consulta Raggruppamento della funzione Lambda.)

```
"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",
};
```

Scansione della tabella DynamoDB

Innanzitutto, crea una funzione async/await chiamata `sendText` per pubblicare un messaggio di testo utilizzando Amazon SNS. `PublishCommand` Quindi, aggiungi uno schema a `try` blocchi che analizza la tabella DynamoDB alla ricerca dei dipendenti che festeggiano oggi il loro anniversario di lavoro, quindi richiama `sendText` la funzione per inviare a questi dipendenti un messaggio di testo. Se si verifica un errore, viene richiamato il `catch` blocco.

Il seguente frammento di codice mostra questa fase. (Per l'esempio completo, consulta Raggruppamento della funzione Lambda.)

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

Raggruppamento della funzione Lambda

Questo argomento descrive come raggruppare i `mylambdafunction.ts` AWS SDK for JavaScript moduli richiesti per questo esempio in un file in bundle chiamato. `index.js`

1. Se non l'hai già fatto, segui questo esempio Attività prerequisite per installare webpack.

> **Note**
>
> Per informazioni sul webpack, consulta. Raggruppa le applicazioni con webpack

2. Esegui quanto segue nella riga di comando per raggruppare il file di questo JavaScript esempio in un file chiamato: `<index.js>`

```
webpack mylambdafunction.ts --mode development --target node --devtool false --
output-library-target umd -o index.js
```

> **Important**
>
> Notate che l'output ha un nome`index.js`. Questo perché le funzioni Lambda devono avere un `index.js` gestore per funzionare.

3. Comprimi il file di output in bundle,`index.js`, in un file ZIP denominato. `mylambdafunction.zip`
4. Carica `mylambdafunction.zip` nel bucket Amazon S3 che hai creato nell'Crea le risorse AWSargomento di questo tutorial.

Distribuire la funzione Lambda

Nella radice del progetto, crea un `lambda-function-setup.ts` file e incolla il contenuto seguente al suo interno.

Sostituisci *BUCKET_NAME* con il nome del bucket Amazon S3 in cui hai caricato la versione ZIP della tua funzione Lambda. Sostituisci *ZIP_FILE_NAME con il nome* o il nome della versione ZIP della tua funzione Lambda. Sostituisci *ROLE* con l'Amazon Resource Number (ARN) del ruolo IAM che hai creato nell'Crea le risorse AWSargomento di questo tutorial. Sostituisci *LAMBDA_FUNCTION_NAME con un nome* per la funzione 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();
```

Immettere quanto segue nella riga di comando per distribuire la funzione Lambda.

```
node lambda-function-setup.ts
```

Questo esempio di codice è disponibile qui. GitHub

Configurare API Gateway per richiamare la funzione Lambda

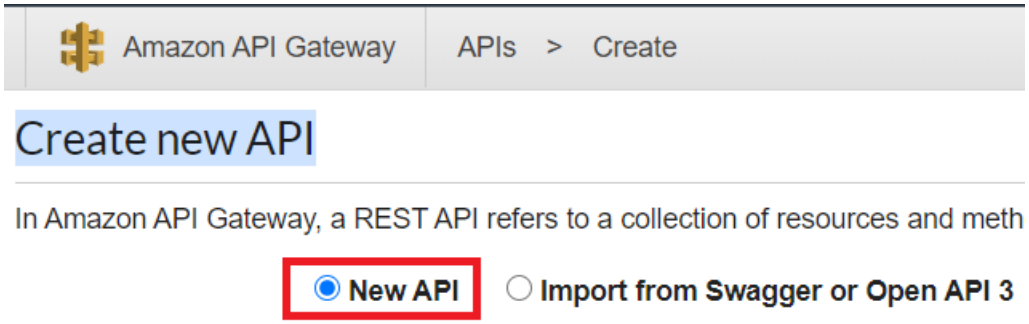
Per creare l'app:

1. Crea la restante API
2. Prova il metodo API Gateway
3. Implementa il metodo API Gateway

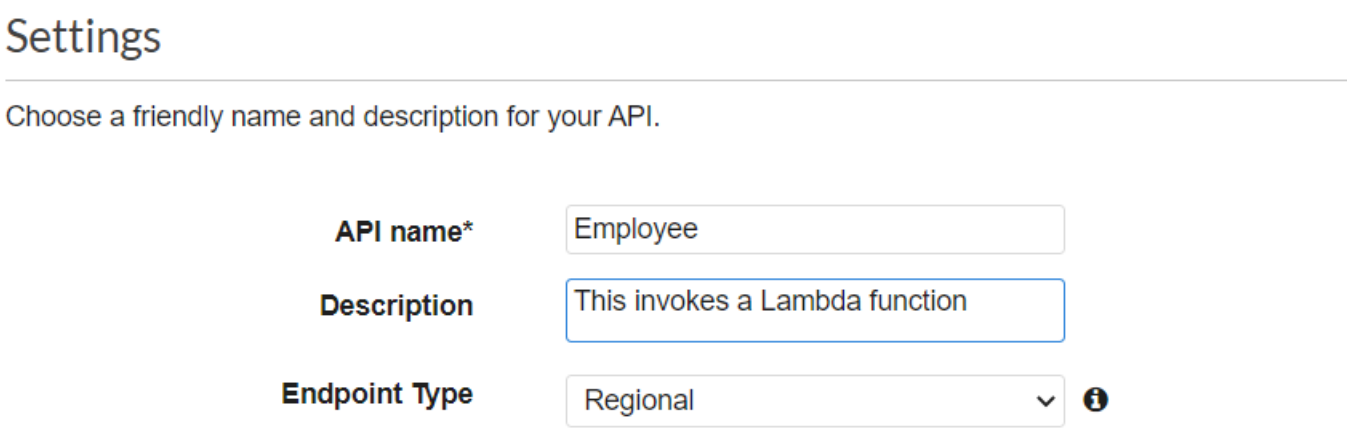
Crea la restante API

Puoi utilizzare la console API Gateway per creare un endpoint REST per la funzione Lambda. Una volta terminata, è possibile richiamare la funzione Lambda utilizzando una chiamata restful.

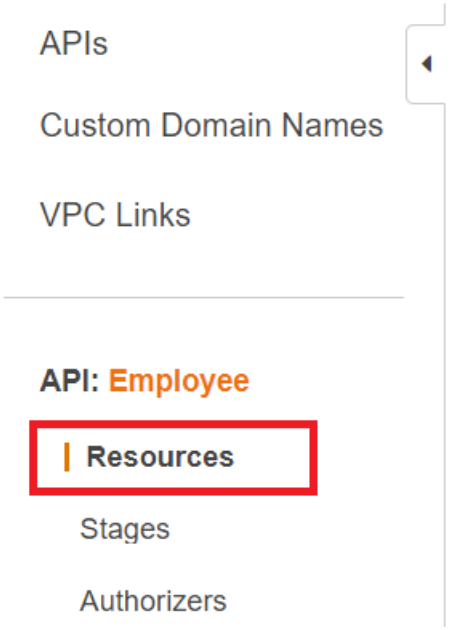
1. Accedi alla console Amazon API Gateway.
2. In Rest API, scegli Build.
3. Seleziona Nuova API.



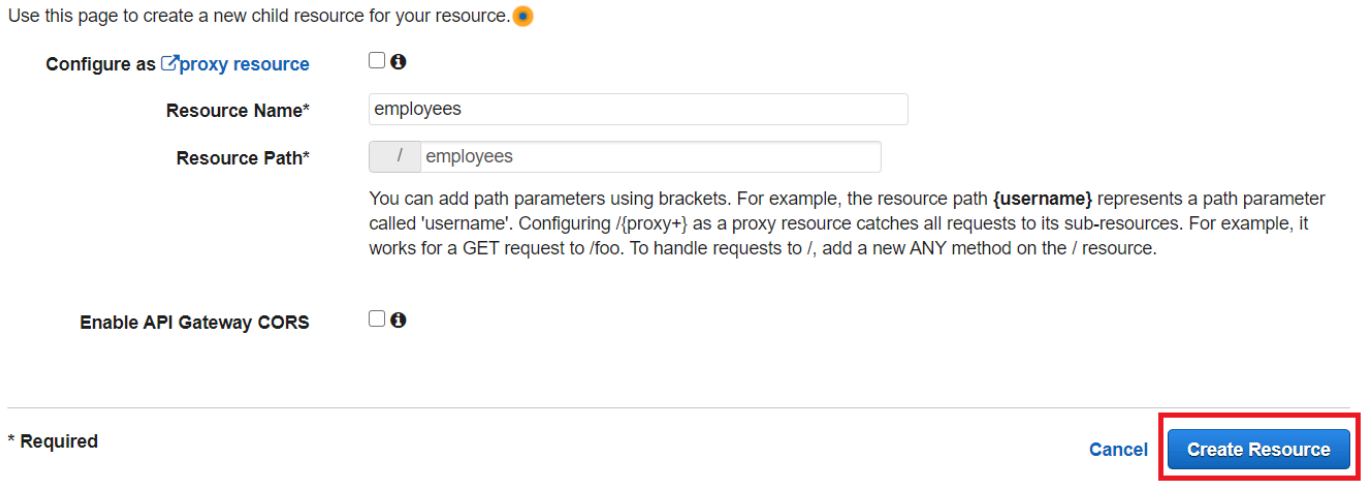
4. Specificate Employee come nome dell'API e fornite una descrizione.



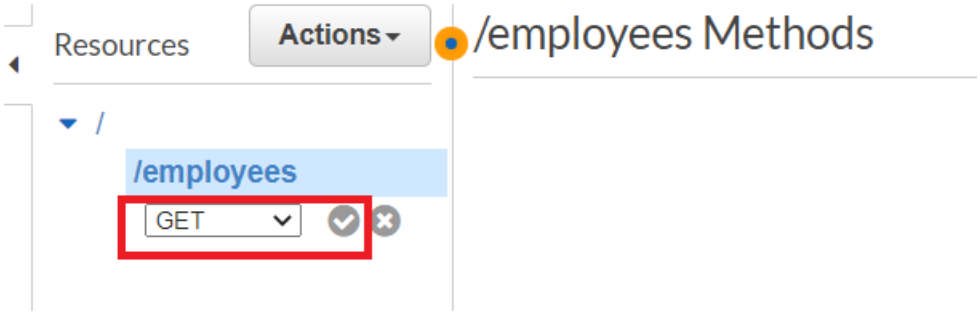
5. Seleziona Create API (Crea API).
6. Scegli Risorse nella sezione Dipendenti.



7. Nel campo del nome, specifica i dipendenti.
8. Selezionare Create Resources (Crea risorse).
9. Dal menu a discesa Azioni, scegli Crea risorse.



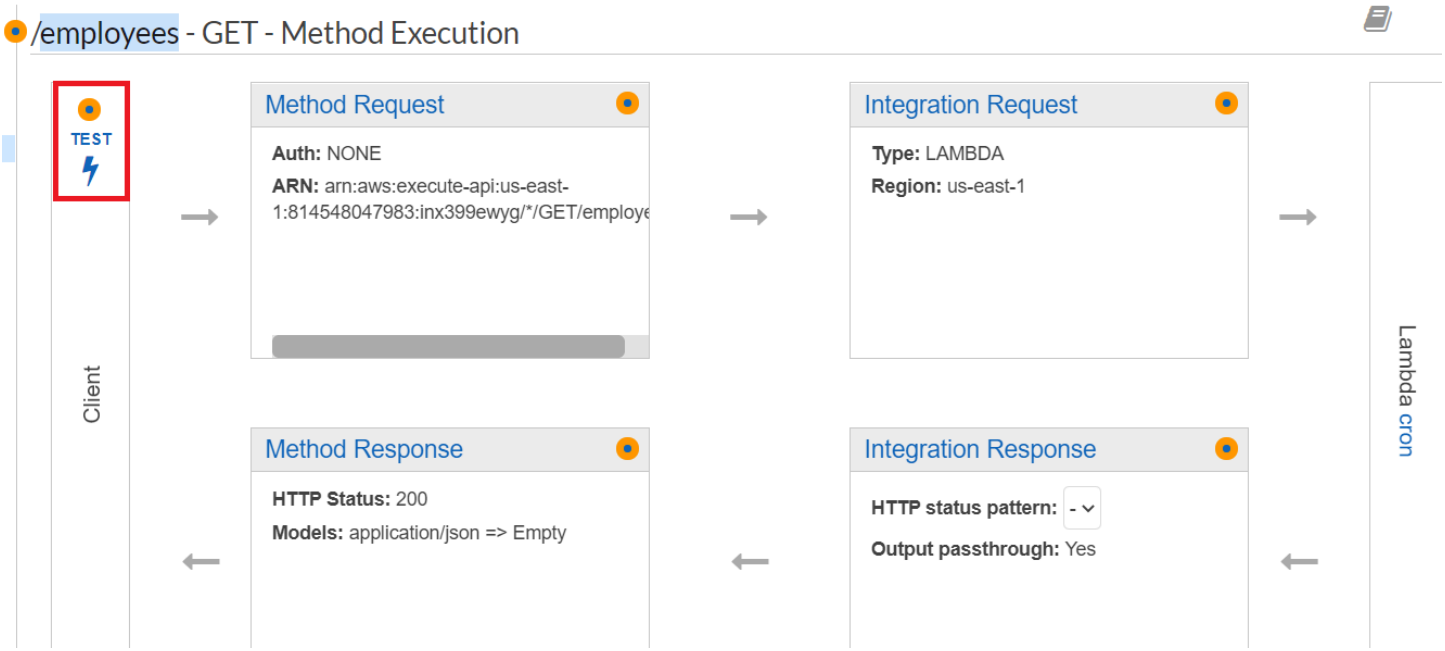
10. Scegli /employees, seleziona Crea metodo dal menu Azioni, quindi seleziona GET dal menu a discesa sotto /employees. Selezionare l'icona del segno di spunta.



11. Scegliete la funzione Lambda e immettete mylambdafunction come nome della funzione Lambda. Selezionare Salva.

Prova il metodo API Gateway

A questo punto del tutorial, puoi testare il metodo API Gateway che richiama la funzione Lambda mylambdafunction. Per testare il metodo, scegliete Test, come illustrato nella figura seguente.

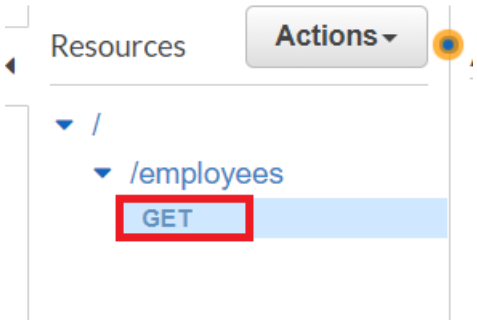


Una volta richiamata la funzione Lambda, è possibile visualizzare il file di registro per visualizzare un messaggio di successo.

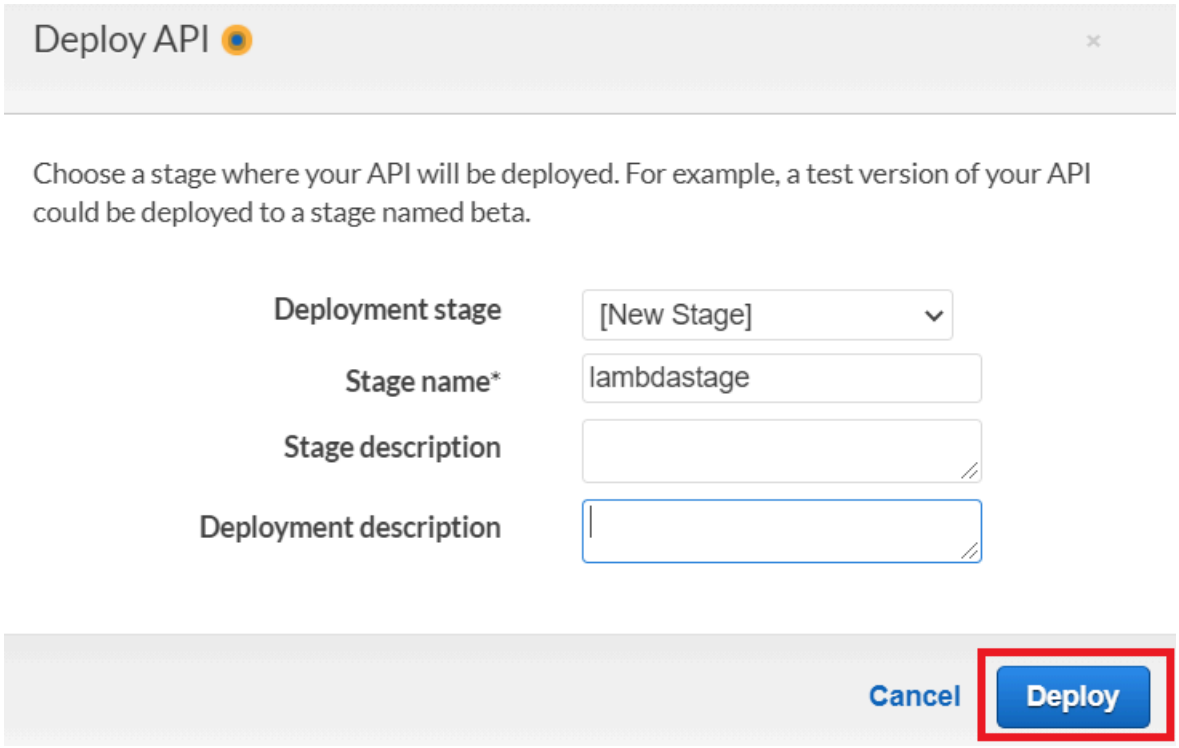
Implementa il metodo API Gateway

Una volta completato il test, puoi implementare il metodo dalla console di Amazon API Gateway.

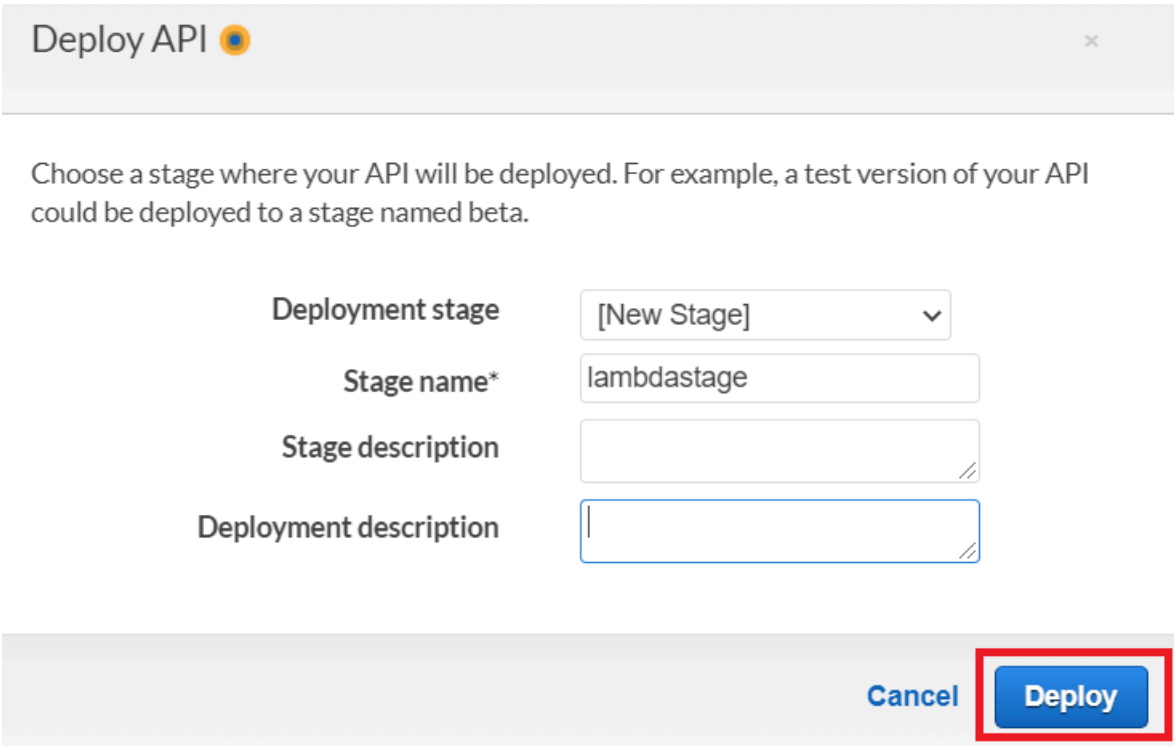
1. Scegli Get.



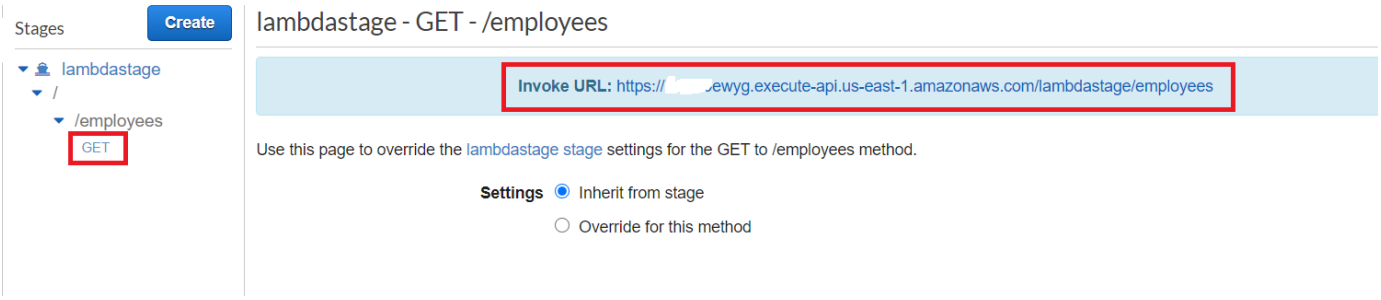
2. Dal menu a discesa Azioni, seleziona Deploy API.



3. Compila il modulo Deploy API e scegli Deploy.



4. Seleziona Salva modifiche.
5. Scegli Get again e nota che l'URL cambia. Questo è l'URL di chiamata che puoi usare per richiamare la funzione Lambda.



Eliminare le risorse

Complimenti! Hai richiamato una funzione Lambda tramite Amazon API Gateway utilizzando il. AWS SDK for JavaScript Come indicato all'inizio di questo tutorial, assicurati di terminare tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti vengano addebitati costi. Puoi farlo eliminando lo AWS CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

1. Apri il file AWS CloudFormationnella console di AWS gestione.

2. Apri la pagina Stacks e seleziona lo stack.
3. Scegli Delete (Elimina).

Creazione di AWS flussi di lavoro serverless utilizzando AWS SDK for JavaScript

È possibile creare un flusso di lavoro AWS serverless utilizzando Step Functions, l'AWSSDK for Java e. AWS Step Functions Ogni fase del flusso di lavoro viene implementata tramite una funzione AWS Lambda. Lambda è un servizio di calcolo che consente di eseguire il codice senza effettuare il provisioning o la gestione di server. Step Functions è un servizio di orchestrazione serverless che consente di combinare funzioni Lambda e altri servizi AWS per la creazione di applicazioni business-critical.

> **Note**
>
> È possibile creare funzioni Lambda in diversi linguaggi di programmazione. In questo tutorial, le funzioni Lambda implementate utilizzando l'API Lambda Java. Per ulteriori informazioni su Lambda, consulta Cos'è Lambda.

In questo tutorial, crei un flusso di lavoro che crea ticket di supporto per un'organizzazione. Ogni fase del flusso di lavoro esegue un'operazione sul ticket. Questo tutorial mostra come utilizzare per JavaScript elaborare i dati del flusso di lavoro. Ad esempio, imparerai come leggere i dati passati al flusso di lavoro, come passare i dati tra i passaggi e come richiamare AWS i servizi dal flusso di lavoro.

Costo di completamento: i AWS servizi inclusi in questo documento sono inclusi nel piano AWS gratuito.

Nota: assicurati di terminare tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti vengano più addebitati costi.

Argomenti

- Attività prerequisite
- Crea le risorse AWS
- Creazione del flusso di lavoro

- Creare le funzioni Lambda
- Aggiungi le funzioni Lambda ai flussi di lavoro
- Esegui il tuo flusso di lavoro utilizzando la console Step Functions
- Eliminare le risorse AWS

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

Crea le risorse AWS

Questo tutorial richiede le seguenti risorse.

- Una tabella Amazon DynamoDB denominata Case con una chiave denominata Id.
- Un ruolo IAM denominato `lambda-support` utilizzato per richiamare le funzioni Lambda. Questo ruolo dispone di politiche che gli consentono di richiamare i servizi Amazon DynamoDB e Amazon Simple Email Service da una funzione Lambda.
- Un ruolo IAM denominato `workflow-support` utilizzato per richiamare il flusso di lavoro.
- Un bucket Amazon S3 per ospitare le funzioni Lambda.

Puoi creare queste risorse manualmente, ma ti consigliamo di effettuare il provisioning di queste risorse utilizzando AWS Cloud Development Kit (AWS CDK) (AWS CDK) come descritto in questo tutorial.

Create le AWS risorse utilizzando AWS CloudFormation

AWS CloudFormation consente di creare ed effettuare il provisioning delle distribuzioni dell'infrastruttura AWS in modo ripetuto e prevedibile. Per ulteriori informazioni su AWS CloudFormation, consulta la AWS CloudFormationGuida per l'utente di .

Per creare lo AWS CloudFormation stack:

1. Installa e configura le AWS CLI seguenti istruzioni contenute nella Guida per l'AWS CLIutente.
2. Create un file denominato `setup.yaml` nella directory principale della cartella del progetto e copiatene il contenuto. GitHub

   > **Note**
   >
   > Il AWS CloudFormation modello è stato generato utilizzando quello AWS CDK disponibile qui GitHub. Per ulteriori informazioni su AWS CDK, consulta la Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK).

3. Esegui il seguente comando dalla riga di comando, sostituendo *`STACK_NAME con un nome`* univoco per lo stack.

   > **Important**
   >
   > Il nome dello stack deve essere univoco all'interno di una regione e di un account. AWS AWS È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

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

   Per ulteriori informazioni sui parametri dei `create-stack` comandi, consultate la guida di riferimento ai AWS CLI comandi e la Guida per l'AWS CloudFormationutente.

Creare le AWS risorse utilizzando la console di gestione di Amazon Web Services;

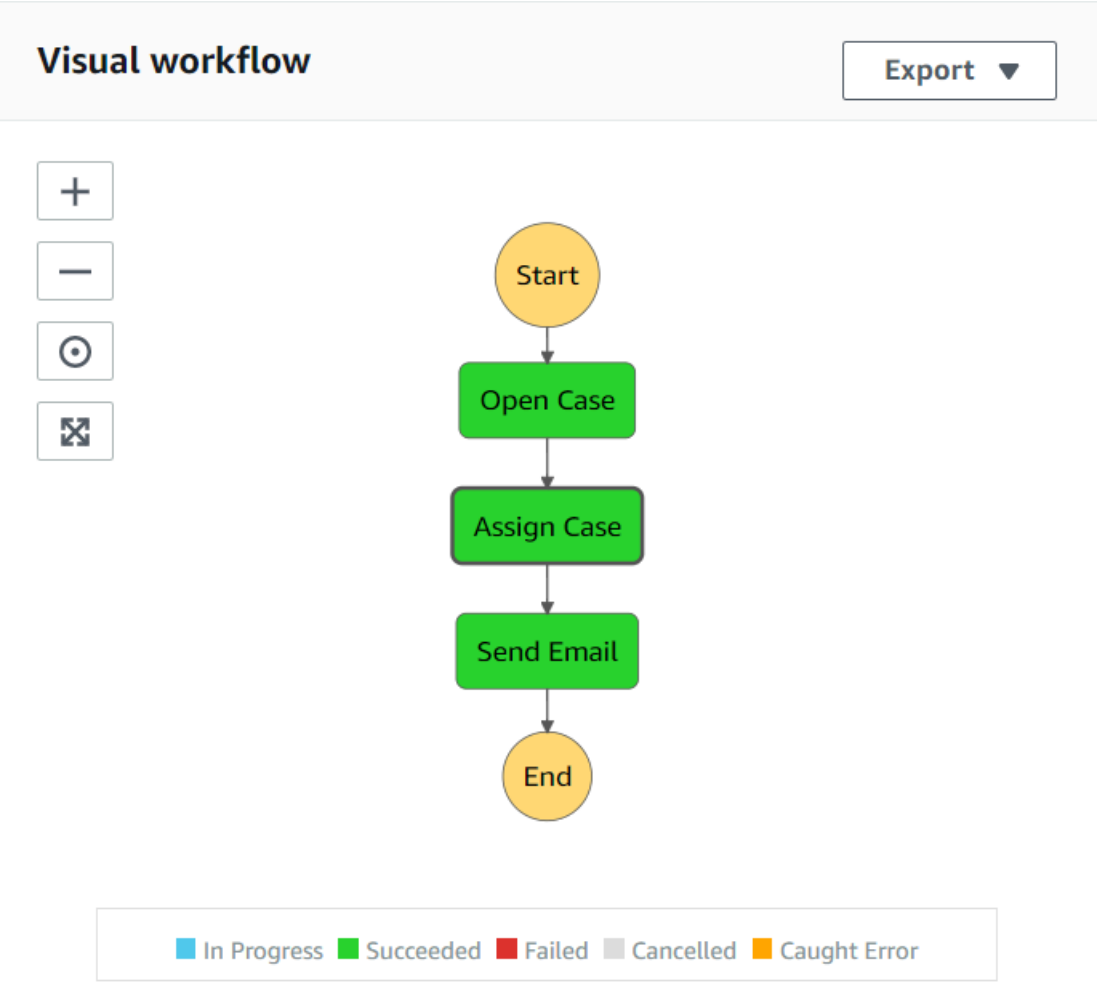
Per creare risorse per l'app nella console, segui le istruzioni nella Guida per l'AWS CloudFormationutente. Usa il modello fornito`setup.yaml`, crea un file denominato e copia il contenuto qui GitHub.

> **Important**
>
> Il nome dello stack deve essere univoco all'interno di una AWS regione e di un AWS account. È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

Visualizza un elenco delle risorse nella console aprendo lo stack sulla AWS CloudFormation dashboard e scegliendo la scheda Risorse. Ti servono per il tutorial.

Creazione del flusso di lavoro

La figura seguente mostra il flusso di lavoro che creerai con questo tutorial.



Di seguito è riportato ciò che accade in ogni fase del flusso di lavoro:

+ Avvia: avvia il flusso di lavoro.

+ Open Case: gestisce il valore dell'ID di un ticket di assistenza trasferendolo al flusso di lavoro.

+ Assegna caso: assegna il caso di supporto a un dipendente e archivia i dati in una tabella DynamoDB.

+ Invia e-mail: invia al dipendente un messaggio e-mail utilizzando Amazon Simple Email Service (Amazon SES) per informarlo dell'esistenza di un nuovo ticket.

+ Fine: interrompe il flusso di lavoro.

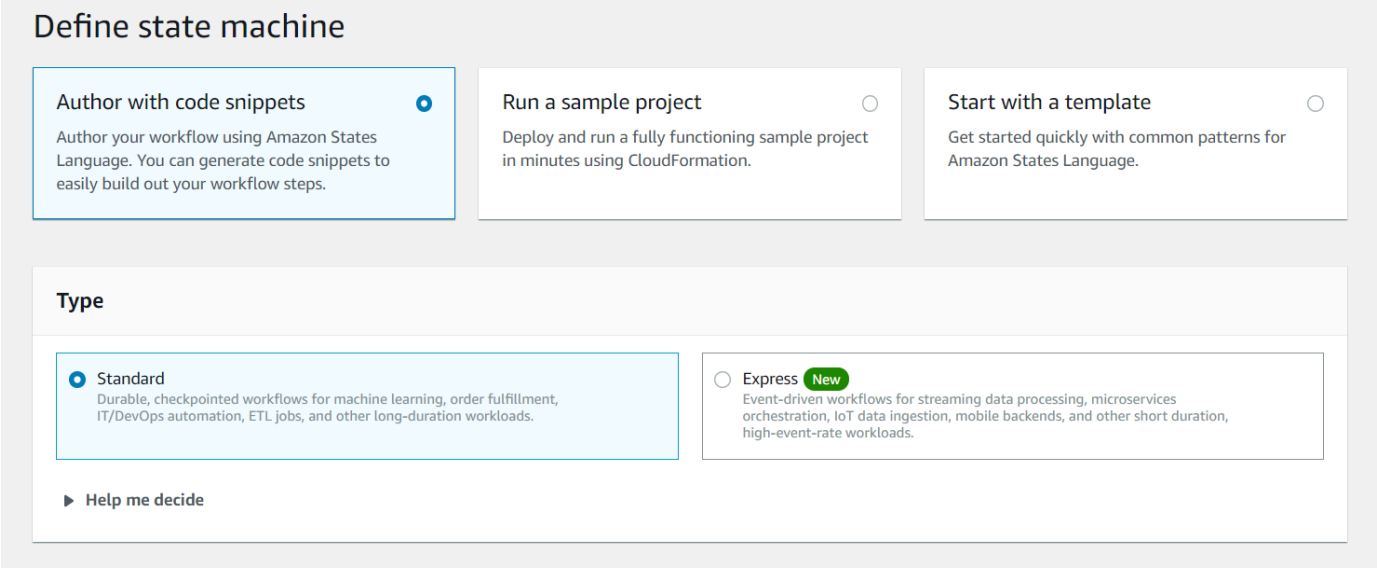
Crea un flusso di lavoro senza server utilizzando le funzioni Step

È possibile creare un flusso di lavoro che elabori i ticket di assistenza. Per definire un flusso di lavoro utilizzando Step Functions, crei un documento Amazon States Language (basato su JSON) per definire la tua macchina a stati. Un documento Amazon States Language descrive ogni passaggio. Dopo aver definito il documento, Step Functions fornisce una rappresentazione visiva del flusso di lavoro. La figura seguente mostra il documento Amazon States Language e la rappresentazione visiva del flusso di lavoro.

I flussi di lavoro possono trasferire dati tra i passaggi. Ad esempio, la fase Open Case elabora un valore ID del caso (passato al flusso di lavoro) e passa tale valore al passaggio Assign Case. Più avanti in questo tutorial, creerai la logica dell'applicazione nella funzione Lambda per leggere ed elaborare i valori dei dati.

Creazione di un flusso di lavoro

1. Apri la console Amazon Web Services.
2. Scegli Create State Machine.
3. Scegliere Author with code snippets (Crea con frammenti di codice). Nell'area Tipo, scegliete Standard.



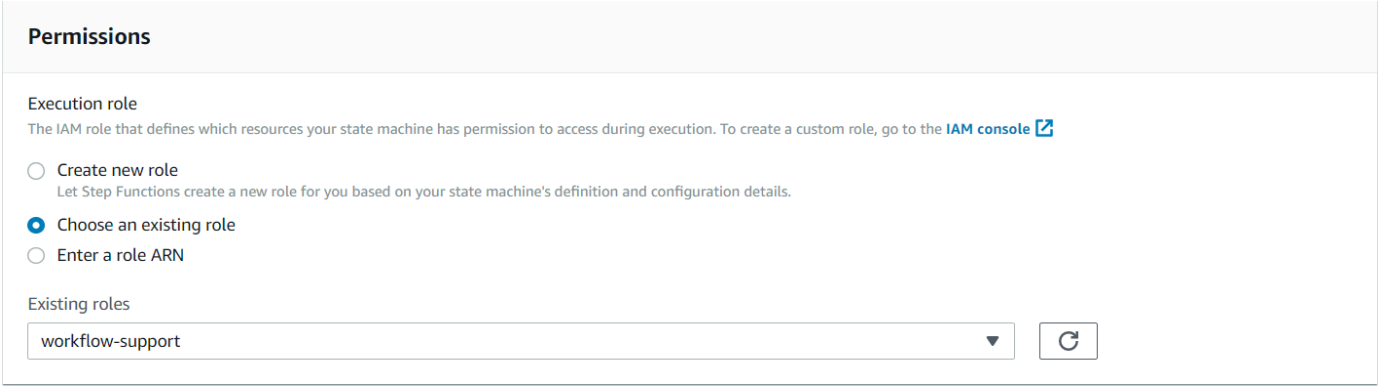
4. Specificare il documento Amazon States Language inserendo il seguente codice.

```
{
```

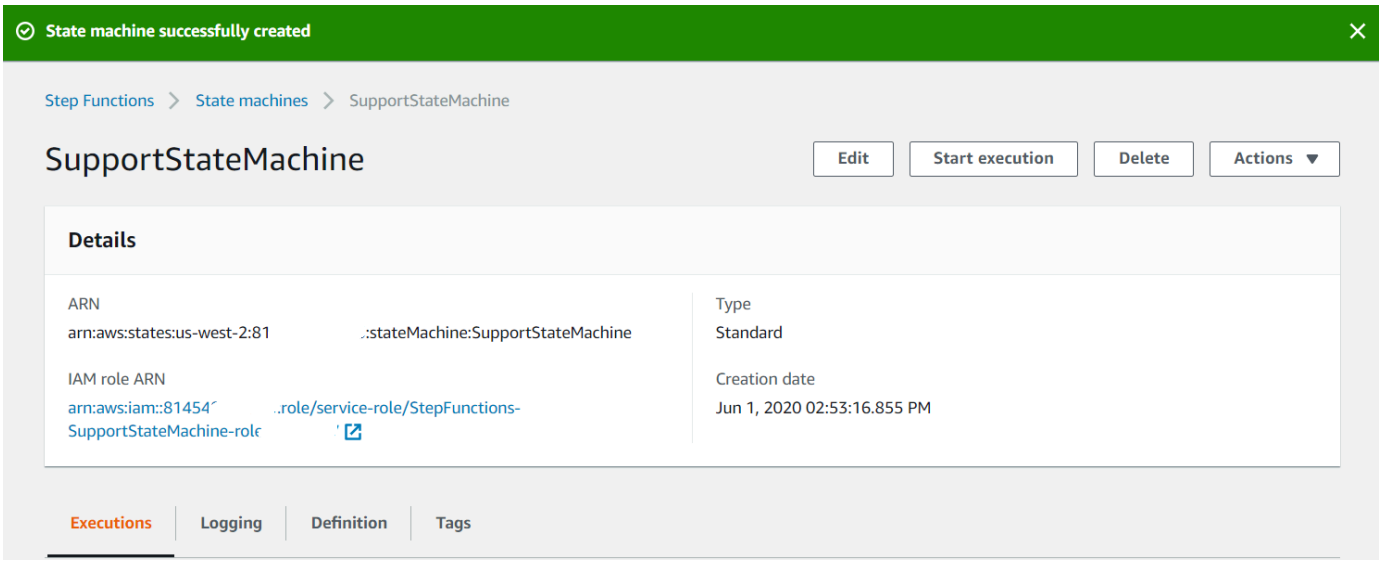
```
"Comment": "A simple AWS Step Functions state machine that automates a call center
 support session.",
"StartAt": "Open Case",
"States": {
"Open Case": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"Next": "Assign Case"
},
"Assign Case": {
"Type": "Task",
"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**
>
> Non preoccuparti degli errori relativi ai valori delle risorse Lambda. Aggiornerai questi valori più avanti in questo tutorial.

5. Seleziona Avanti.
6. Nel campo del nome, inserisci SupportStateMachine.
7. Nella sezione Autorizzazione, scegli Scegli un ruolo esistente.
8. Scegli workflow-support (il ruolo IAM che hai creato).



9. Scegli Create state machine (Crea macchina a stati). Viene visualizzato un messaggio che indica che la macchina a stati è stata creata correttamente.



Creare le funzioni Lambda

Utilizza l'API Lambda runtime per creare le funzioni Lambda. In questo esempio, ci sono tre fasi del flusso di lavoro che corrispondono ciascuna a ciascuna funzione Lambda.

Crea queste funzioni Lambda, come descritto nelle seguenti sezioni:

- Funzione GetID Lambda- Utilizzato come primo passaggio del flusso di lavoro che elabora il valore dell'ID del ticket.
- Classe AddItem Lambda- Utilizzato come seconda fase del flusso di lavoro che assegna il ticket a un dipendente e archivia i dati in un database DynamoDB.
- sendemail Classe Lambda- Utilizzato come terza fase del flusso di lavoro che utilizza Amazon SES per inviare un messaggio e-mail al dipendente per informarlo del ticket.

Funzione GetID Lambda

Crea una funzione Lambda che restituisca il valore dell'ID del ticket passato alla seconda fase del flusso di lavoro.

```
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);
   }
};
```

Immetti quanto segue nella riga di comando per utilizzare webpack per raggruppare il file in un file denominato. `index.js`

```
webpack getid.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

Quindi comprimi `index.js` in un `ZIP` nome di file. `getid.js.zip` Carica il `ZIP` file nel bucket Amazon S3 che hai creato nell'argomento di questo esempio.

Questo esempio di codice è disponibile qui. GitHub

Classe AddItem Lambda

Crea una funzione Lambda che seleziona un dipendente a cui assegnare il ticket, quindi archivia i dati del ticket in una tabella DynamoDB denominata 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);
  }
};
```

Immettete quanto segue nella riga di comando per utilizzare webpack per raggruppare il file in un file denominato. `index.js`

```
webpack additem.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

Quindi comprimi `index.js` in un `ZIP` nome di file. `additem.js.zip` Carica il `ZIP` file nel bucket Amazon S3 che hai creato nell'argomento di questo esempio.

Questo esempio di codice è disponibile qui. GitHub

sendemail Classe Lambda

Crea una funzione Lambda che invii un'e-mail per avvisarli del nuovo ticket. Viene utilizzato l'indirizzo e-mail trasmesso dal secondo passaggio.

```
// Load the required clients and commands.
const { SendEmailCommand } = require ( "@aws-sdk/client-ses" );
const { sesClient } = require ( "../libs/sesClient" );

exports.handler = async (event) => {
  // Enter a sender email address. This address must be verified.
  const senderEmail = "SENDER_EMAIL"
  const sender = "Sender Name <" + senderEmail + ">";

  // AWS Step Functions passes the  employee's email to the event.
  // This address must be verified.
  const recepient = event.S;

  // The subject line for the email.
  const subject = "New case";

  // The email body for recipients with non-HTML email clients.
  const body_text =
    "Hello,\r\n" + "Please check the database for new ticket assigned to you.";

  // The HTML body of the email.
```

```
  const body_html = `<html><head></head><body><h1>Hello!</h1><p>Please check the
 database for new ticket assigned to you.</p></body></html>`;

  // The character encoding for the email.
  const charset = "UTF-8";
  var params = {
    Source: sender,
    Destination: {
      ToAddresses: [recepient],
    },
    Message: {
      Subject: {
        Data: subject,
        Charset: charset,
      },
      Body: {
        Text: {
          Data: body_text,
          Charset: charset,
        },
        Html: {
          Data: body_html,
          Charset: charset,
        },
      },
    },
  };
  try {
    const data = await sesClient.send(new SendEmailCommand(params));
    console.log(data);
  } catch (err) {
    console.error(err);
  }
};
```

Immettere quanto segue nella riga di comando per utilizzare webpack per raggruppare il file in un file denominato. `index.js`

```
webpack sendemail.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

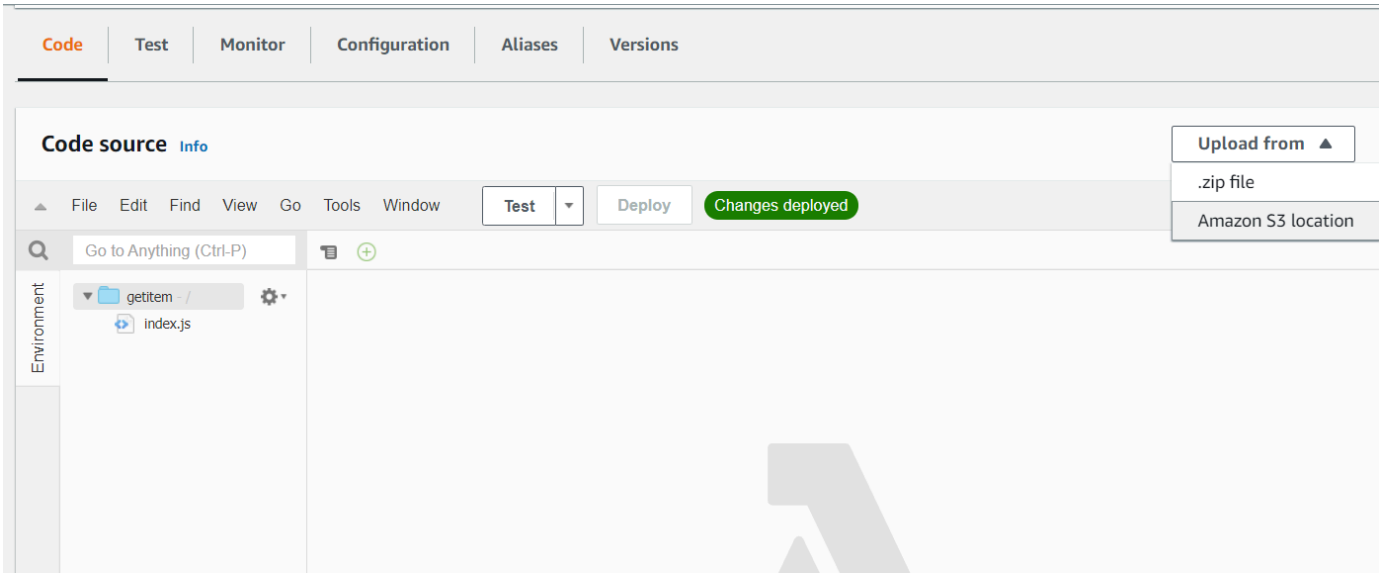
Quindi comprimi `index.js` in un `ZIP` nome di file. `sendemail.js.zip` Carica il `ZIP` file nel bucket Amazon S3 che hai creato nell'argomento di questo esempio.

Questo esempio di codice è disponibile qui. GitHub

Implementa le funzioni Lambda

Per distribuire la funzione `getid` Lambda:

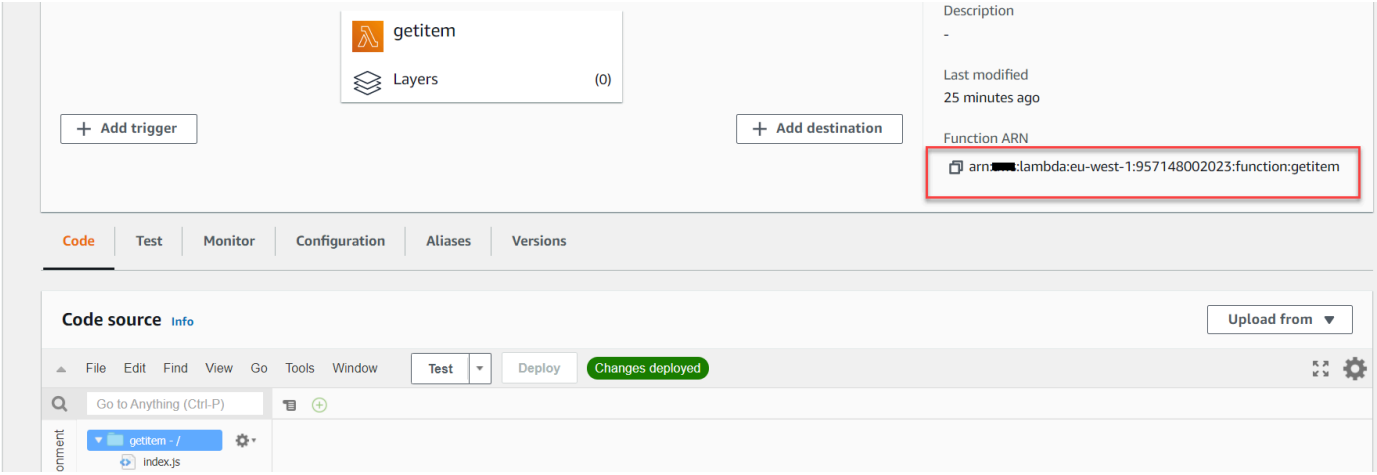
1. Apri la console Lambda nella console di Amazon Web Services.
2. Selezionare Create function (Crea funzione).
3. Scegli Author from scratch (Crea da zero).
4. Nella sezione Informazioni di base, inserisci getid come nome.
5. In Runtime, scegliete Node.js 14x.
6. Scegli Usa un ruolo esistente, quindi scegli lambda-support (il ruolo IAM che hai creato in).
7. Scegli Crea funzione.
8. scegli Carica da - Posizione Amazon S3.
9. Scegli Carica, scegli Carica da - posizione Amazon S3 e inserisci l'URL del link Amazon S3.



10. Selezionare Salva.
11. Ripeti questa procedura per additem.js.zip e sendemail.js.zip alle nuove funzioni Lambda. Al termine, avrai a disposizione tre funzioni Lambda a cui potrai fare riferimento nel documento Amazon States Language.

Aggiungi le funzioni Lambda ai flussi di lavoro

1. Apri la console Lambda. Nota che puoi visualizzare il valore Lambda Amazon Resource Name (ARN) nell'angolo in alto a destra.



2. Copia il valore e incollalo nella fase 1 del documento Amazon States Language, che si trova nella console Step Functions.
3. Aggiorna i passaggi Resource for the Assign Case e Send Email. Ecco come agganciare le funzioni Lambda create utilizzando l'AWSSDK for Java a un flusso di lavoro creato utilizzando Step Functions.

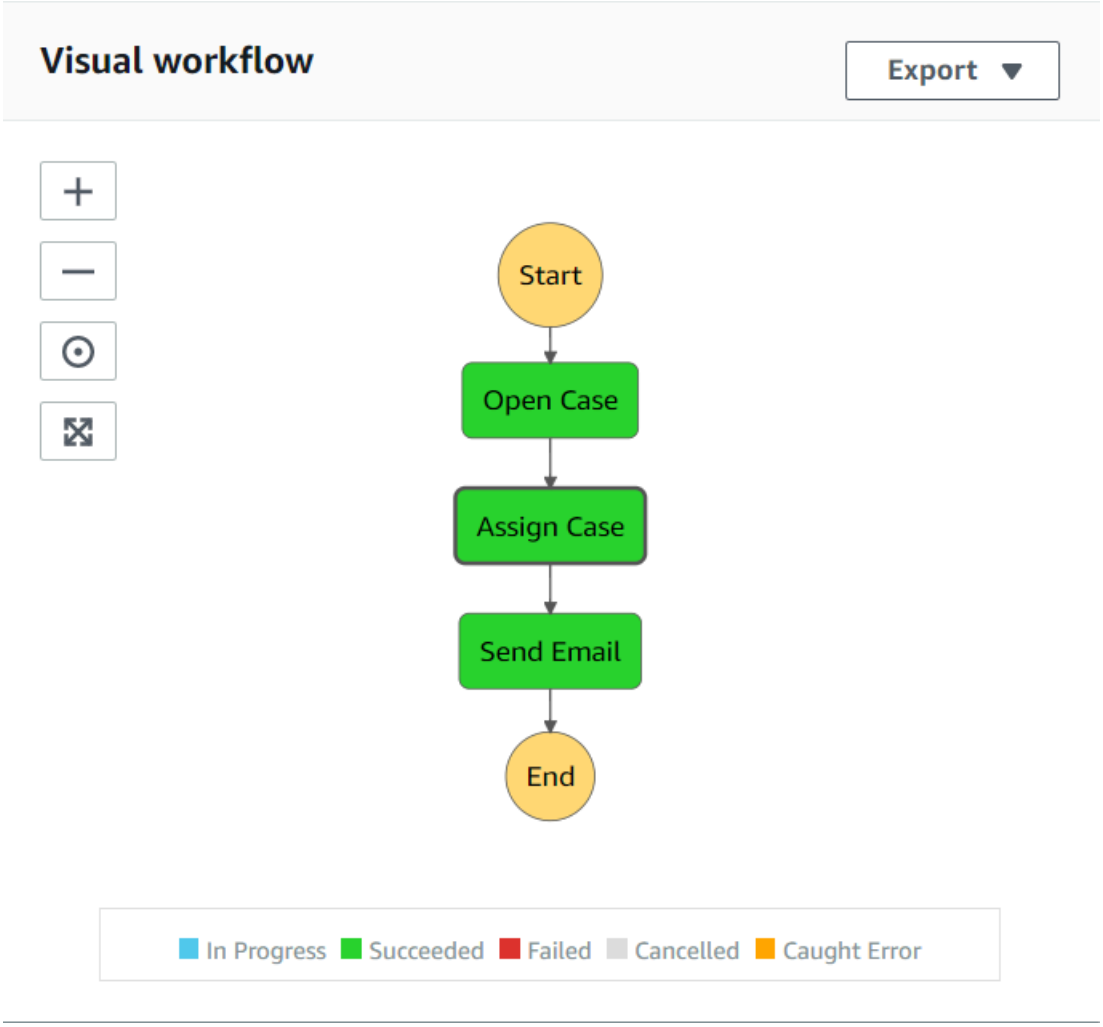
Esegui il tuo flusso di lavoro utilizzando la console Step Functions

È possibile richiamare il flusso di lavoro sulla console Step Functions. Un'esecuzione riceve l'input JSON. Per questo esempio, puoi passare i seguenti dati JSON al flusso di lavoro.

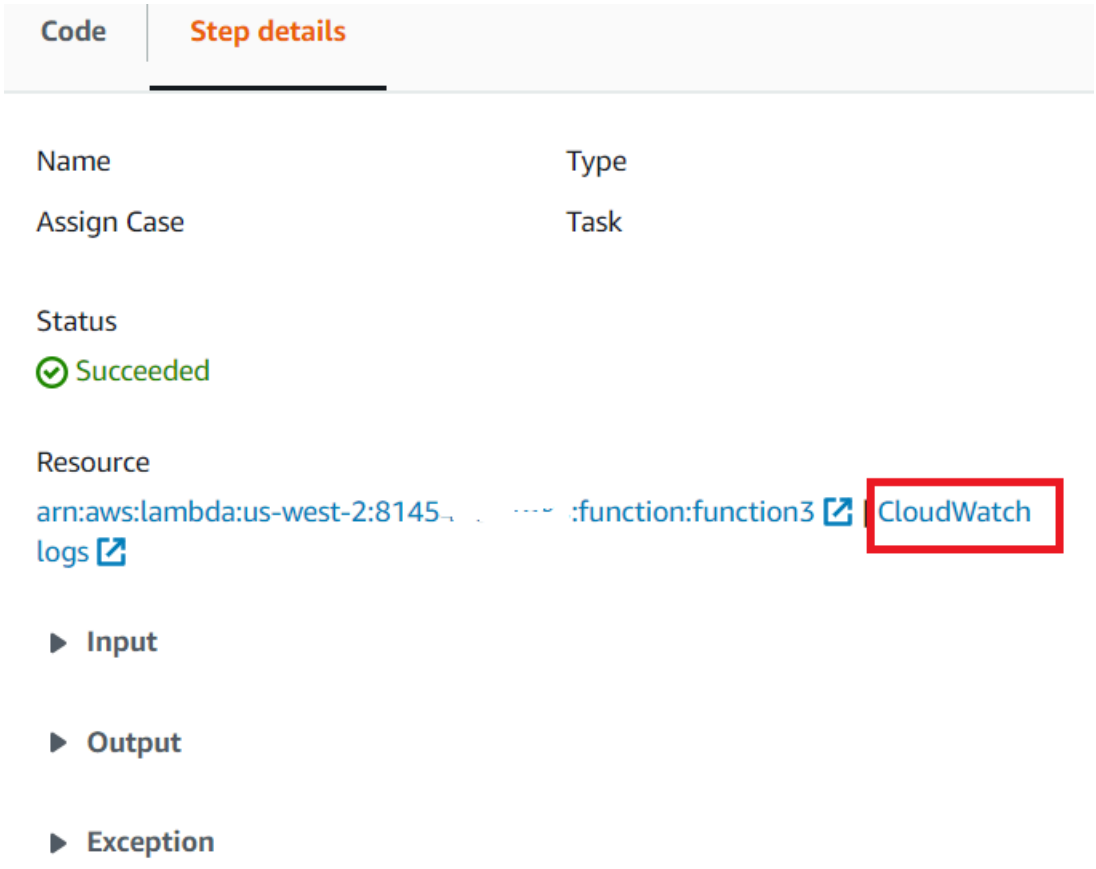
```
{
"inputCaseID": "001"
}
```

Per eseguire il flusso di lavoro:

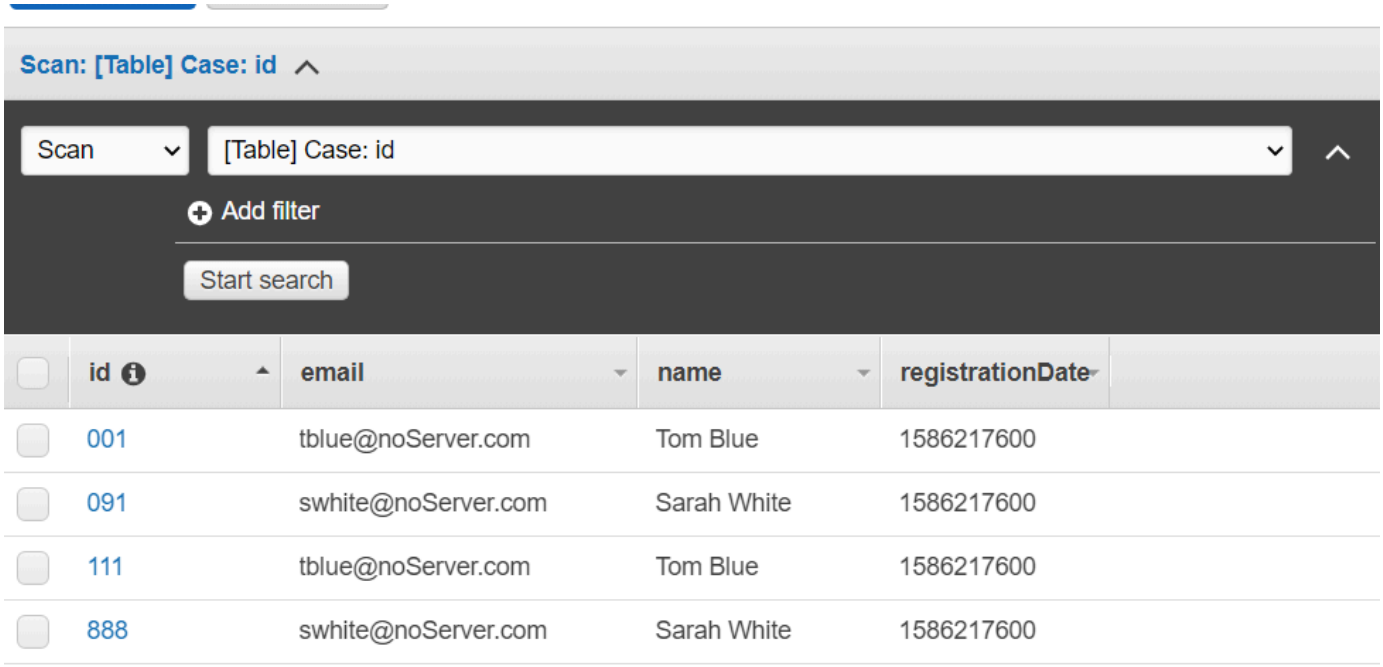
1. Nella console Step Functions, scegli Avvia esecuzione.
2. Nella sezione Input, passa i dati JSON. Visualizza il flusso di lavoro. Quando ogni passaggio è completato, diventa verde.



3. Se il passaggio diventa rosso, si è verificato un errore. È possibile fare clic sul passaggio e visualizzare i registri accessibili dal lato destro.



Al termine del flusso di lavoro, è possibile visualizzare i dati nella tabella DynamoDB.



id	email	name	registrationDate
001	tblue@noServer.com	Tom Blue	1586217600
091	swhite@noServer.com	Sarah White	1586217600
111	tblue@noServer.com	Tom Blue	1586217600
888	swhite@noServer.com	Sarah White	1586217600

Eliminare le risorse AWS

Congratulazioni, avete creato un flusso di lavoro AWS serverless utilizzando l'AWSSDK for Java. Come indicato all'inizio di questo tutorial, assicurati di terminare tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti vengano addebitati costi. Puoi farlo eliminando lo AWS CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

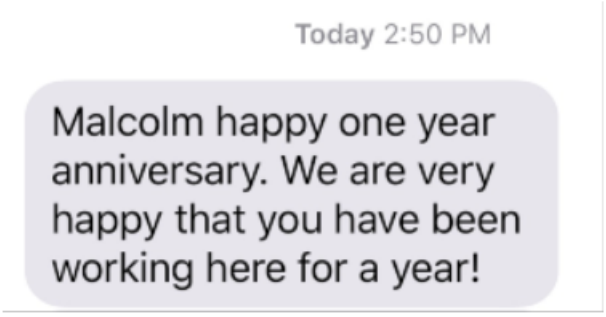
1. Apri il file AWS CloudFormationnella console di AWS gestione.
2. Apri la pagina Stacks e seleziona lo stack.
3. Scegli Delete (Elimina).

Creazione di eventi pianificati per eseguire AWS Lambda funzioni

Puoi creare un evento pianificato che richiami una AWS Lambda funzione utilizzando un Amazon CloudWatch Event. È possibile configurare un CloudWatch evento per utilizzare un'espressione cron per pianificare quando viene richiamata una funzione Lambda. Ad esempio, puoi pianificare un CloudWatch evento per richiamare una funzione Lambda ogni giorno della settimana.

AWS Lambdaè un servizio di elaborazione che consente di eseguire codice senza fornire o gestire server. È possibile creare funzioni Lambda in diversi linguaggi di programmazione. Per ulteriori informazioni su AWS Lambda, consulta Che cos'è AWS Lambda.

In questo tutorial, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi servizi AWS per eseguire un caso d'uso specifico. Ad esempio, supponiamo che un'organizzazione invii un messaggio di testo mobile ai propri dipendenti per congratularsi con loro in occasione del primo anniversario, come illustrato in questa illustrazione.

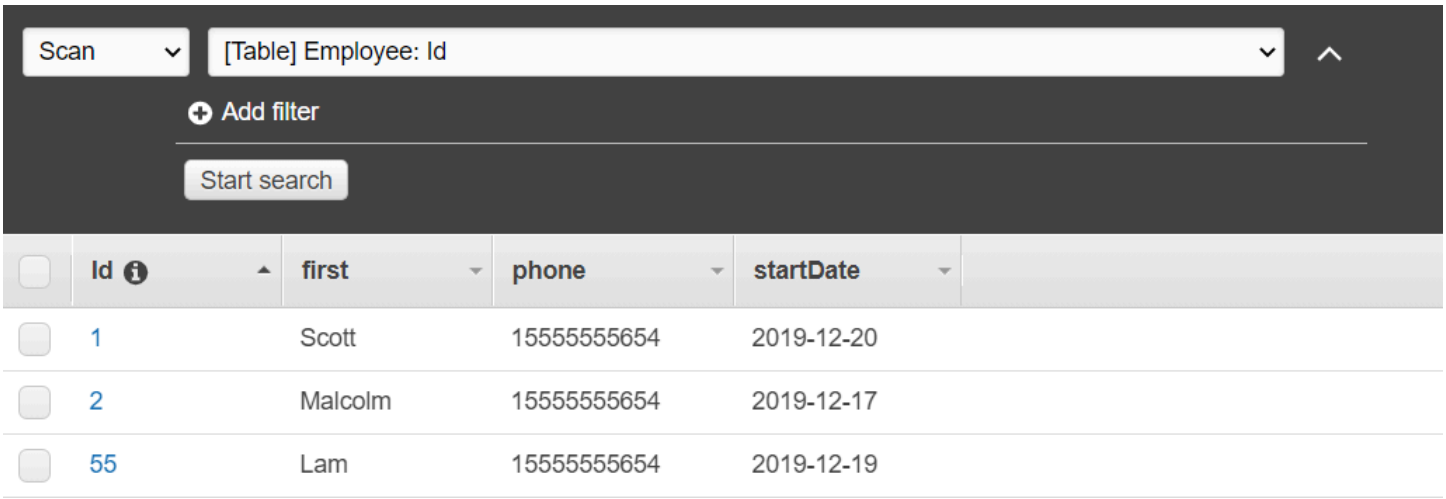


Il completamento di questo tutorial richiede circa 20 minuti.

Questo tutorial mostra come utilizzare la JavaScript logica per creare una soluzione che esegua questo caso d'uso. Ad esempio, imparerai a leggere un database per determinare quali dipendenti hanno raggiunto la data del primo anniversario, come elaborare i dati e inviare un messaggio di testo, il tutto utilizzando una funzione Lambda. Quindi imparerai come usare un'espressione cron per richiamare la funzione Lambda ogni giorno della settimana.

Questo AWS tutorial utilizza una tabella Amazon DynamoDB denominata Employee che contiene questi campi.

- id: la chiave primaria per la tabella.
- firstName: nome del dipendente.
- telefono: numero di telefono del dipendente.
- StartDate: data di inizio del dipendente.



> **Important**
>
> Costo di completamento: i AWS servizi inclusi in questo documento sono inclusi nel piano AWS gratuito. Tuttavia, assicurati di interrompere tutte le risorse dopo aver completato questo tutorial per assicurarti che non ti venga addebitato alcun costo.

Per creare l'app:

1. Prerequisiti completi
2. Crea le risorse AWS

3. Preparare lo script del browser
4. Crea e carica la funzione Lambda
5. Implementa la funzione Lambda
6. Esegui l'app
7. Elimina le risorse

Attività prerequisite

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node.js e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

Crea le risorse AWS

Questo tutorial richiede le seguenti risorse.

- Una tabella Amazon DynamoDB denominata Employee con una chiave denominata Id e i campi mostrati nell'illustrazione precedente. Assicurati di inserire i dati corretti, incluso un telefono cellulare valido con cui testare questo caso d'uso. Per ulteriori informazioni, consulta Creare una tabella.
- Un ruolo IAM con autorizzazioni allegate per eseguire funzioni Lambda.
- Un bucket Amazon S3 per ospitare la funzione Lambda.

Puoi creare queste risorse manualmente, ma ti consigliamo di effettuare il provisioning di queste risorse utilizzando il metodo AWS CloudFormation descritto in questo tutorial.

Crea le AWS risorse utilizzando AWS CloudFormation

AWS CloudFormation consente di creare ed effettuare il provisioning delle distribuzioni dell'infrastruttura AWS in modo ripetuto e prevedibile. Per ulteriori informazioni su AWS CloudFormation, consulta la AWS CloudFormationGuida per l'utente di .

Per creare lo AWS CloudFormation stack usando: AWS CLI

1. Installa e configura le AWS CLI seguenti istruzioni contenute nella Guida per l'AWS CLIutente.
2. Create un file denominato `setup.yaml` nella directory principale della cartella del progetto e copiatene il contenuto. GitHub

   > **Note**
   >
   > Il AWS CloudFormation modello è stato generato utilizzando quello AWS CDK disponibile qui GitHub. Per ulteriori informazioni su AWS CDK, consulta la Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK).

3. Esegui il seguente comando dalla riga di comando, sostituendo *`STACK_NAME con un nome`* univoco per lo stack.

   > **Important**
   >
   > Il nome dello stack deve essere univoco all'interno di una regione e di un account. AWS AWS È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

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

   Per ulteriori informazioni sui parametri dei `create-stack` comandi, consultate la guida di riferimento ai AWS CLI comandi e la Guida per l'AWS CloudFormationutente.

   Visualizza un elenco delle risorse nella console aprendo lo stack sulla AWS CloudFormation dashboard e scegliendo la scheda Risorse. Ti servono per il tutorial.
4. Quando lo stack viene creato, utilizzare il AWS SDK for JavaScript per popolare la tabella DynamoDB, come descritto in. Compila la tabella DynamoDB

Compila la tabella DynamoDB

Per compilare la tabella, create innanzitutto una directory denominata`libs`, in essa create un file denominato `dynamoClient.js` e incollate il contenuto sottostante al suo interno.

```
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 };
```

Questo codice è disponibile qui su. GitHub

Quindi, crea un file denominato `populate-table.js` nella directory principale della cartella del tuo progetto e copia il contenuto qui GitHub dentro. Per uno degli elementi, sostituisci il valore della `phone` proprietà con un numero di cellulare valido nel formato E.164 e il valore della `startDate` con la data odierna.

Esegui il comando seguente dalla riga di comando.

```
node populate-table.js
```

```
const {
BatchWriteItemCommand } = require( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require(  "./libs/dynamoClient" );
// Set the parameters.
const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
```

```
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Questo codice è disponibile qui GitHub.

Creazione della funzione AWS Lambda

Configurazione dell'SDK

Importa innanzitutto i moduli e i comandi richiesti AWS SDK for JavaScript (v3): `ScanCommand` DynamoDB `DynamoDBClient` e `SNSClient` il comando Amazon SNS. `PublishCommand` Sostituisci *`REGION con Region.`* AWS Quindi calcola la data odierna e assegnala a un parametro. Quindi crea i parametri per `ScanCommand` .Replace *`TABLE_NAME con il nome`* della tabella che hai creato nella sezione di questo esempio. Crea le risorse AWS

Il seguente frammento di codice mostra questa fase. (Per l'esempio completo, consulta Raggruppamento della funzione Lambda.)

```
"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",
};
```

Scansione della tabella DynamoDB

Per prima cosa crea una funzione async/await chiamata `sendText` a pubblicare un messaggio di testo utilizzando Amazon SNS. `PublishCommand` Quindi, aggiungi uno schema a `try` blocchi che analizza la tabella DynamoDB alla ricerca dei dipendenti che festeggiano oggi il loro anniversario di lavoro, quindi richiama `sendText` la funzione per inviare a questi dipendenti un messaggio di testo. Se si verifica un errore, viene richiamato il `catch` blocco.

Il seguente frammento di codice mostra questa fase. (Per l'esempio completo, consulta Raggruppamento della funzione Lambda.)

```
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);
  }
};
```

Raggruppamento della funzione Lambda

Questo argomento descrive come raggruppare i `mylambdafunction.js` AWS SDK for JavaScript moduli richiesti per questo esempio in un file in bundle chiamato. `index.js`

1. Se non l'hai già fatto, segui questo esempio Attività prerequisite per installare webpack.

   > **Note**
   >
   > Per informazioni sul webpack, consulta. Raggruppa le applicazioni con webpack

2. Esegui quanto segue nella riga di comando per raggruppare il file di questo JavaScript esempio in un file chiamato: `<index.js>`

   ```
   webpack mylamdbafunction.js --mode development --target node --devtool false --
   output-library-target umd -o index.js
   ```

> **Important**
>
> Notate che l'output ha un nome`index.js`. Questo perché le funzioni Lambda devono avere un `index.js` gestore per funzionare.

3. Comprimi il file di output in bundle,`index.js`, in un file ZIP denominato. `my-lambda-function.zip`
4. Carica `mylambdafunction.zip` nel bucket Amazon S3 che hai creato nell'Crea le risorse AWSargomento di questo tutorial.

Ecco il codice completo dello script del browser per. `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);
  }
};
```

Distribuire la funzione Lambda

Nella radice del progetto, crea un `lambda-function-setup.js` file e incolla il contenuto seguente al suo interno.

Sostituisci *`BUCKET_NAME`* con il nome del bucket Amazon S3 in cui hai caricato la versione ZIP della tua funzione Lambda. Sostituisci *`ZIP_FILE_NAME con il nome`* o il nome della versione ZIP della tua funzione Lambda. Sostituisci *`IAM_ROLE_ARN`* con l'Amazon Resource Number (ARN)

del ruolo IAM che hai creato nell'argomento di questo tutorial. Crea le risorse AWS Sostituisci *LAMBDA_FUNCTION_NAME con un nome* per la funzione 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();
```

Immettere quanto segue nella riga di comando per distribuire la funzione Lambda.

```
node lambda-function-setup.js
```

Questo esempio di codice è disponibile qui. GitHub

Configura CloudWatch per richiamare le funzioni Lambda

CloudWatch Per configurare l'invocazione delle funzioni Lambda:

1. Aprire la pagina Functions (Funzioni) nella console Lambda.
2. Scegli la funzione Lambda.
3. Under Designer, scegliere Add trigger (Aggiungi trigger).
4. Imposta il tipo di trigger su CloudWatch EventBridgeEvents/.
5. Per Regola, scegli Crea una nuova regola.
6. Inserisci il nome della regola e la descrizione della regola.
7. Per il tipo di regola, seleziona Espressione di pianificazione.
8. Nel campo Schedule expression, inserisci un'espressione cron. Ad esempio, cron (0) 12? * DAL LUNEDÌ AL VENERDÌ (*).
9. Scegli Aggiungi.

> **Note**
>
> Per ulteriori informazioni, consulta Uso di Lambda con CloudWatch gli eventi.

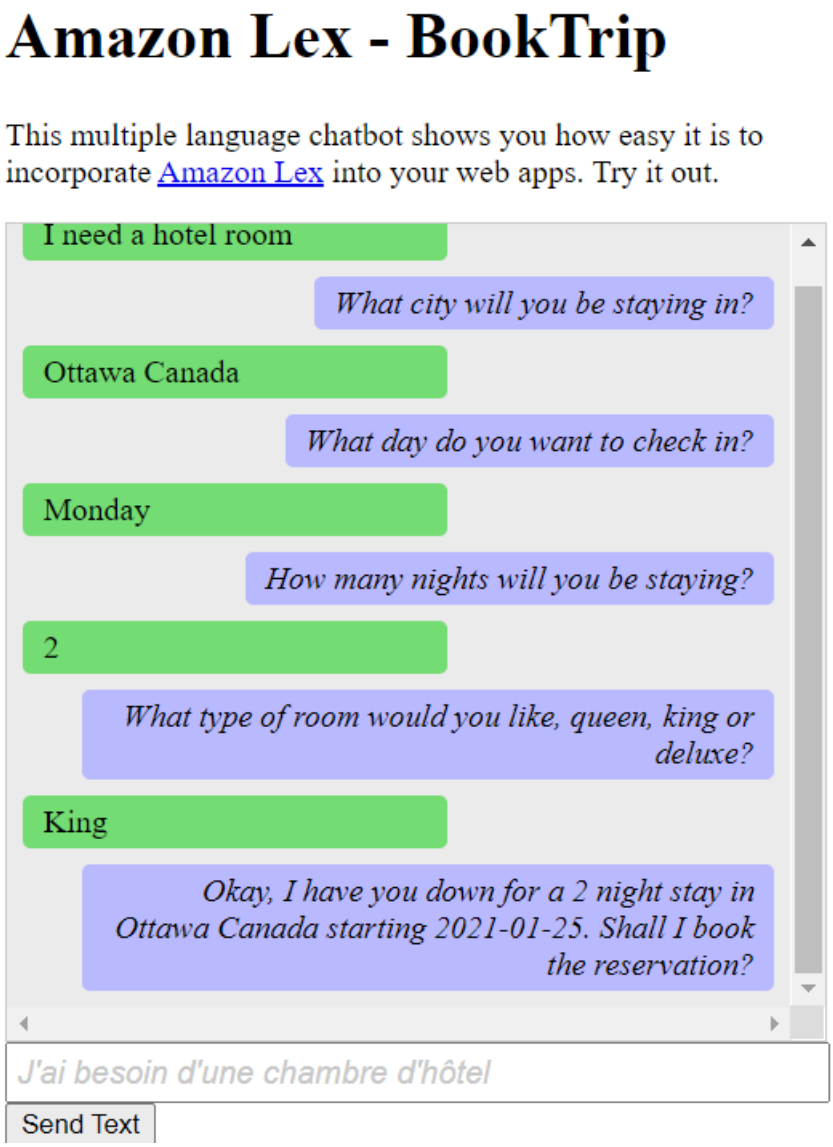
Eliminare le risorse

Complimenti! Hai richiamato una funzione Lambda tramite eventi pianificati di CloudWatch Amazon utilizzando. AWS SDK for JavaScript Come indicato all'inizio di questo tutorial, assicurati di terminare tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti vengano addebitati costi. Puoi farlo eliminando lo AWS CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

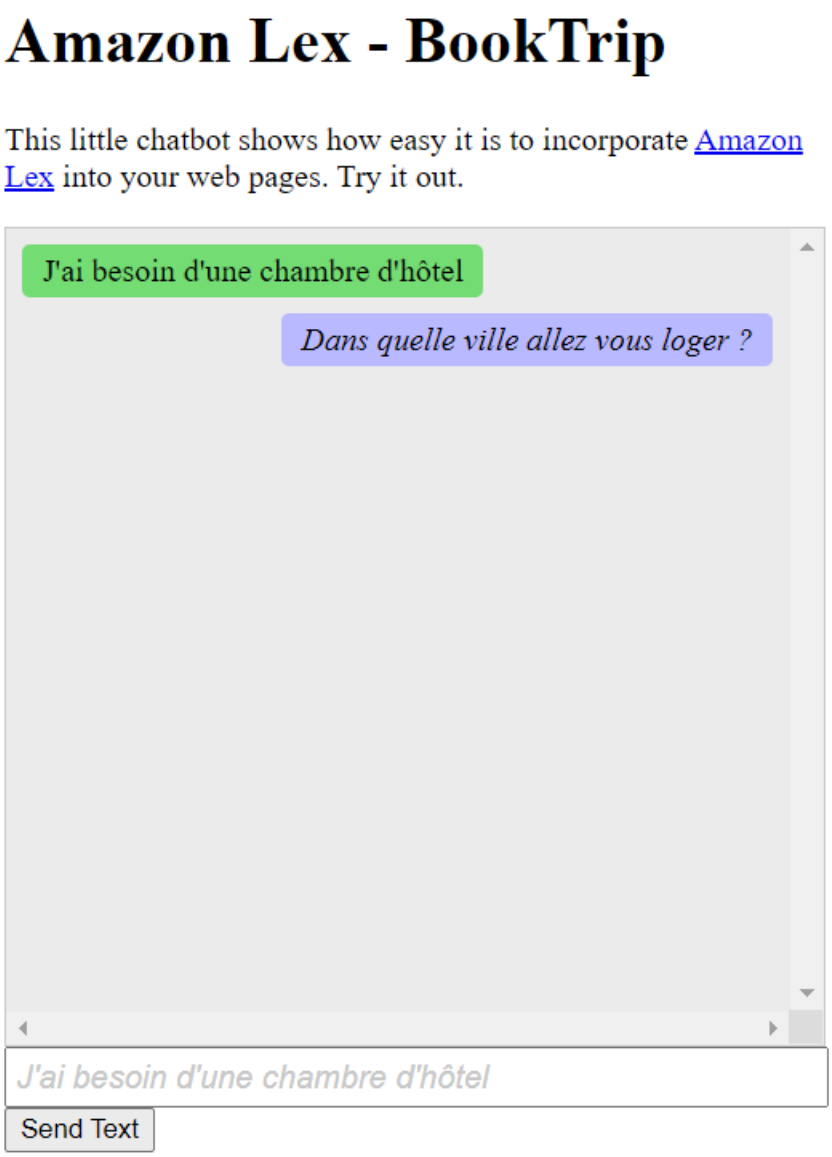
1. Apri la AWS CloudFormation console.
2. Nella pagina Stacks, seleziona lo stack.
3. Scegli Delete (Elimina).

Creazione di un chatbot Amazon Lex

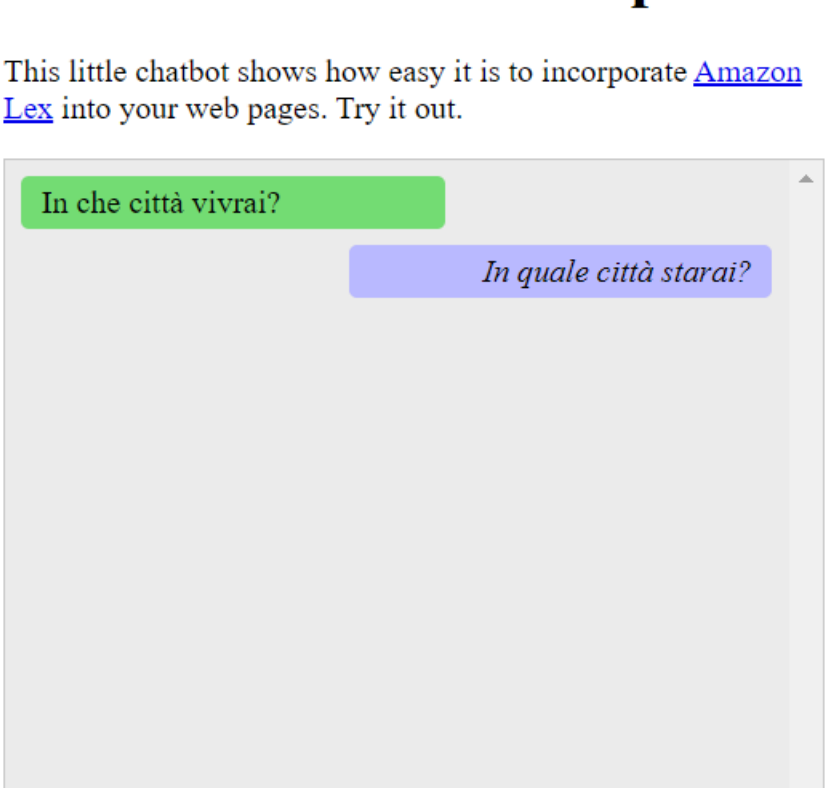
Puoi creare un chatbot Amazon Lex all'interno di un'applicazione Web per coinvolgere i visitatori del tuo sito Web. Un chatbot di Amazon Lex è una funzionalità che esegue conversazioni in chat online con gli utenti senza fornire un contatto diretto con una persona. Ad esempio, l'illustrazione seguente mostra un chatbot Amazon Lex che coinvolge un utente nella prenotazione di una camera d'albergo.



Il chatbot Amazon Lex creato in questo AWS tutorial è in grado di gestire più lingue. Ad esempio, un utente che parla francese può inserire testo in francese e ottenere una risposta in francese.



Allo stesso modo, un utente può comunicare con il chatbot di Amazon Lex in italiano.



Questo AWS tutorial ti guida nella creazione di un chatbot Amazon Lex e nella sua integrazione in un'applicazione Web Node.js. La AWS SDK for JavaScript (v3) viene utilizzata per richiamare questi servizi: AWS

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

Costo di completamento: i AWS servizi inclusi in questo documento sono inclusi nel piano AWSgratuito.

Nota: assicurati di interrompere tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti venga addebitato alcun costo.

Per creare l'app:

1. Prerequisiti
2. Effettuare il provisioning delle risorse

3. Crea un chatbot Amazon Lex
4. Crea il codice HTML
5. Crea lo script del browser
6. Fasi successive

Prerequisiti

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questo esempio utilizza ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Crea le risorse AWS

Questo tutorial richiede le seguenti risorse.

- Un ruolo IAM non autenticato con autorizzazioni allegate per:
  - Amazon Comprehend
  - Amazon Translate
  - Amazon Lex

Puoi creare queste risorse manualmente, ma ti consigliamo di effettuare il provisioning di queste risorse utilizzando AWS CloudFormation quanto descritto in questo tutorial.

Crea le AWS risorse utilizzando AWS CloudFormation

AWS CloudFormation consente di creare ed effettuare il provisioning delle distribuzioni dell'infrastruttura AWS in modo ripetuto e prevedibile. Per ulteriori informazioni su AWS CloudFormation, consulta la AWS CloudFormationGuida per l'utente di .

Per creare lo AWS CloudFormation stack usando: AWS CLI

1. Installa e configura le AWS CLI seguenti istruzioni contenute nella Guida per l'AWS CLIutente.
2. Create un file denominato `setup.yaml` nella directory principale della cartella del progetto e copiatene il contenuto. GitHub

   > **Note**
   >
   > Il AWS CloudFormation modello è stato generato utilizzando quello AWS CDK disponibile qui GitHub. Per ulteriori informazioni su AWS CDK, consulta la Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK).

3. Esegui il seguente comando dalla riga di comando, sostituendo *`STACK_NAME con un nome`* univoco per lo stack.

   > **Important**
   >
   > Il nome dello stack deve essere univoco all'interno di una regione e di un account. AWS AWS È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

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

   Per ulteriori informazioni sui parametri dei `create-stack` comandi, consultate la guida di riferimento ai AWS CLI comandi e la Guida per l'AWS CloudFormationutente.

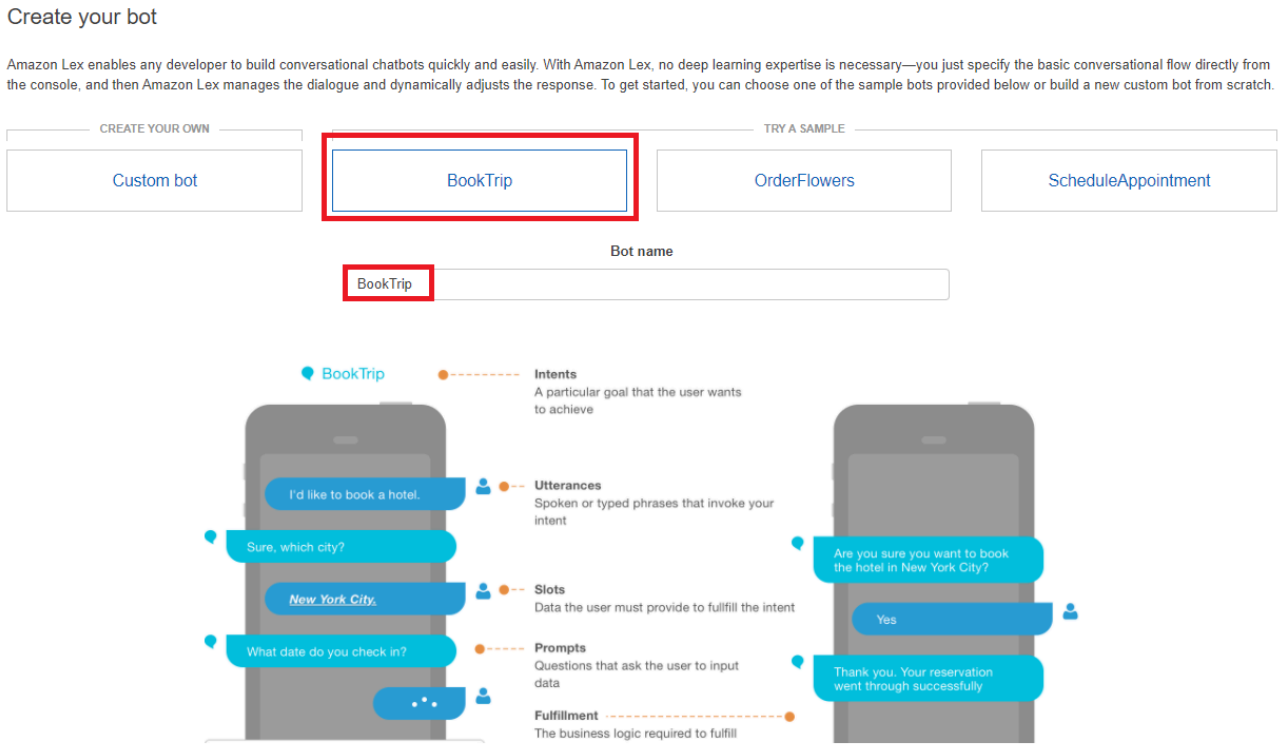
   Per visualizzare le risorse create, apri la console Amazon Lex, scegli lo stack e seleziona la scheda Risorse.

Creazione di un bot Amazon Lex

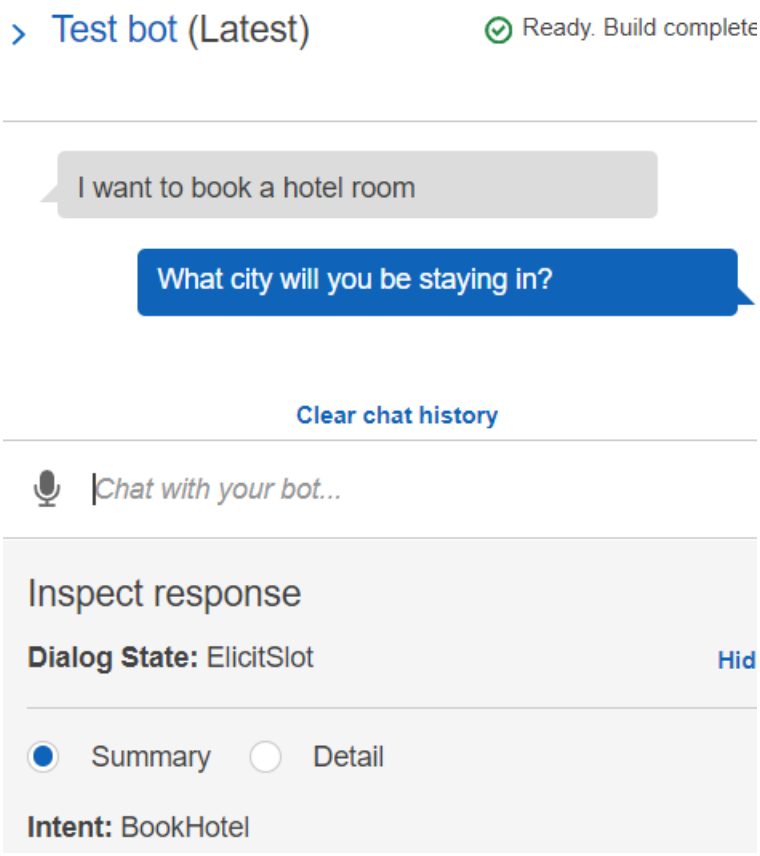
> **Important**
>
> Usa la versione 1 della console Amazon Lex per creare il bot. Questo esempio non funziona con i bot creati utilizzando V2.

Il primo passaggio consiste nel creare un chatbot Amazon Lex utilizzando la console di gestione di Amazon Web Services. In questo esempio, viene utilizzato l'BookTripesempio Amazon Lex. Per ulteriori informazioni, consulta Book Trip.

- Accedi alla console di gestione di Amazon Web Services e apri la console Amazon Lex su Amazon Web Services Console.
- Nella pagina Bot, scegli Crea.
- Scegli BookTripblueprint (lascia il nome BookTrippredefinito del bot).



- Inserisci le impostazioni predefinite e scegli Crea (la console mostra il BookTripbot). Nella scheda Editor, esamina i dettagli degli intenti preconfigurati.
- Esegui il test del bot nella finestra di prova. Inizia il test digitando Voglio prenotare una camera d'albergo.



- Scegliete Pubblica e specificate un nome alias (questo valore vi servirà quando usate ilAWS SDK for JavaScript).

> **Note**
>
> Devi fare riferimento al nome del bot e all'alias del bot nel codice JavaScript .

Crea l'HTML

Crea un file denominato `index.html`. Copia e incolla il codice seguente in`index.html`. Questo riferimento HTML`main.js`. Questa è una versione in bundle di index.js, che include i AWS SDK for JavaScript moduli richiesti. Creerai questo file inCrea l'HTML. `index.html`anche riferimenti`style.css`, che aggiungono gli stili.

```
<!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>
```

Questo codice è disponibile anche qui GitHub.

Crea lo script del browser

Crea un file denominato `index.js`. Copia e incolla il codice seguente in`index.js`. Importa i AWS SDK for JavaScript moduli e i comandi richiesti. Crea clienti per Amazon Lex, Amazon Comprehend e Amazon Translate. Sostituisci *`AWSREGION`* con Region e *`IDENTITY_POOL_ID con l'ID del pool`* di identità che hai creato in. Crea le risorse AWS Per recuperare l'ID del pool di identità, apri il pool di identità nella console Amazon Cognito, scegli Modifica pool di identità e scegli Codice di esempio nel menu laterale. L'ID del pool di identità viene visualizzato in rosso nella console.

Innanzitutto, crea una `libs` directory, crea gli oggetti client di servizio richiesti creando tre file`comprehendClient.js`,`lexClient.js`, `etranslateClient.js`. Incolla il codice appropriato di seguito in ciascuno di essi e sostituisci *`REGION`* e *`IDENTITY_POOL_ID`* in ogni file.

> **Note**
>
> Usa l'ID del pool di identità di Amazon Cognito in cui hai creato. Crea le AWS risorse utilizzando 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 };
```

Questo codice è disponibile qui. GitHub .

Quindi, crea un `index.js` file e incolla il codice seguente al suo interno.

Sostituisci BOT_ALIAS e BOT_NAME rispettivamente con l'alias e il nome del tuo bot Amazon Lex e USER_ID con un ID utente. La funzione asincrona esegue le seguenti operazioni: `createResponse`

- Prende il testo immesso dall'utente nel browser e utilizza Amazon Comprehend per determinarne il codice della lingua.
- Prende il codice della lingua e usa Amazon Translate per tradurre il testo in inglese.
- Prende il testo tradotto e utilizza Amazon Lex per generare una risposta.
- Pubblica la risposta nella pagina del browser.

```
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;
```

Questo codice è disponibile qui GitHub. .

Ora usa webpack per raggruppare i AWS SDK for JavaScript moduli `index.js` and in un unico file,. `main.js`

1. Se non l'hai già fatto, segui questo esempio Prerequisiti per installare webpack.

   > **Note**
   >
   > Per informazioni sul webpack, consulta. Raggruppa le applicazioni con webpack

2. Esegui quanto segue nella riga di comando per raggruppare il file di questo JavaScript esempio in un file chiamato: `main.js`

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

Passaggi successivi

Complimenti! Hai creato un'applicazione Node.js che utilizza Amazon Lex per creare un'esperienza utente interattiva. Come indicato all'inizio di questo tutorial, assicurati di terminare tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti vengano addebitati costi. Puoi farlo eliminando lo AWS CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

1. Apri la AWS CloudFormation console.
2. Nella pagina Stacks, seleziona lo stack.
3. Scegli Elimina.

Per altri esempi AWS inter-service, consulta AWS SDK for JavaScript Esempi interservice.

Creazione di un'applicazione di messaggistica di esempio

Puoi creare un'AWSapplicazione che invia e recupera messaggi utilizzando Amazon AWS SDK for JavaScript Simple Queue Service (Amazon SQS). I messaggi vengono archiviati in una coda FIFO (first in, first out) che garantisce che l'ordine dei messaggi sia coerente. Ad esempio, il primo messaggio archiviato nella coda è il primo messaggio letto dalla coda.

> **Note**
>
> Per ulteriori informazioni su Amazon SQS, consulta What is Amazon Simple Queue Service?

In questo tutorial, crei un'applicazione Node.js denominata AWS Messaging.

Costo di completamento: i AWS servizi inclusi in questo documento sono inclusi nel piano AWS gratuito.

Nota: assicurati di interrompere tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti venga addebitato alcun costo.

Per creare l'app:

1. Prerequisiti
2. Effettuare il provisioning delle risorse
3. Comprendi il flusso di lavoro
4. Crea il codice HTML
5. Crea lo script del browser
6. Fasi successive

Prerequisiti

Per configurare ed eseguire questo esempio, è necessario completare queste attività:

- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa i moduli richiesti AWS SDK for JavaScript e di terze parti. Segui le istruzioni su GitHub.
- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWSSDK e agli strumenti.

> **Important**
>
> Questo esempio utilizza ECMAScript6 (ES6). Ciò richiede la versione 13.x o successiva di Node.js. Per scaricare e installare la versione più recente di Node.js, consulta Node.js downloads. .
> Tuttavia, se preferisci utilizzare la sintassi CommonJS, fai riferimento a. JavaScript Sintassi ES6/CommonJS

Crea le risorse AWS

Questo tutorial richiede le seguenti risorse.

- Un ruolo IAM non autenticato con autorizzazioni per Amazon SQS.
- Una coda Amazon SQS FIFO denominata Message.FIFO: per informazioni sulla creazione di una coda, consulta Creazione di una coda Amazon SQS.

Puoi creare queste risorse manualmente, ma ti consigliamo di effettuare il provisioning di queste risorse utilizzando () come descritto in questo tutorial. AWS CloudFormation AWS CloudFormation

> **Note**
>
> AWS CloudFormationSi tratta di un framework di sviluppo software che consente di definire le risorse delle applicazioni cloud. Per ulteriori informazioni, consulta la AWS CloudFormationGuida per l'utente.

Crea le AWS risorse utilizzando il AWS CloudFormation

AWS CloudFormation consente di creare ed effettuare il provisioning delle distribuzioni dell'infrastruttura AWS in modo ripetuto e prevedibile. Per ulteriori informazioni su AWS CloudFormation, consulta la AWS CloudFormationGuida per l'utente di .

Per creare lo AWS CloudFormation stack utilizzando: AWS CLI

1. Installa e configura le AWS CLI seguenti istruzioni contenute nella Guida per l'AWS CLIutente.
2. Create un file denominato `setup.yaml` nella directory principale della cartella del progetto e copiatene il contenuto. GitHub

   > **Note**
   >
   > Il AWS CloudFormation modello è stato generato utilizzando quello AWS CDK disponibile qui GitHub. Per ulteriori informazioni su AWS CDK, consulta la Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK).

3. Esegui il seguente comando dalla riga di comando, sostituendo *`STACK_NAME con un nome`* univoco per lo stack.

   > **Important**
   >
   > Il nome dello stack deve essere univoco all'interno di una regione e di un account. AWS AWS È possibile specificare fino a 128 caratteri e sono consentiti numeri e trattini.

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

   Per ulteriori informazioni sui parametri dei `create-stack` comandi, consultate la guida di riferimento ai AWS CLI comandi e la Guida per l'AWS CloudFormationutente.

   Per visualizzare le risorse create, apri AWS CloudFormation nella console di AWS gestione, scegli lo stack e seleziona la scheda Risorse.

Comprendi l'applicazione AWS di messaggistica

Per inviare un messaggio a una coda SQS, inserisci il messaggio nell'applicazione e scegli Invia.

Dopo l'invio del messaggio, l'applicazione visualizza il messaggio.

Puoi scegliere Purge per eliminare i messaggi dalla coda di Amazon SQS. Ciò si traduce in una coda vuota e nessun messaggio viene visualizzato nell'applicazione.

Di seguito viene descritto come l'applicazione gestisce un messaggio:

- L'utente seleziona il proprio nome e immette il messaggio, quindi invia il messaggio, avviando la funzione. `pushMessage`
- `pushMessage`recupera l'URL della coda Amazon SQS, quindi invia un messaggio con un valore ID messaggio univoco (un GUID), il testo del messaggio e l'utente alla coda Amazon SQS.
- `pushMessage`recupera i messaggi dalla coda Amazon SQS, estrae l'utente e il messaggio per ogni messaggio e visualizza i messaggi.
- L'utente può eliminare i messaggi, eliminandoli dalla coda di Amazon SQS e dall'interfaccia utente.

Crea la pagina HTML

Ora create i file HTML necessari per l'interfaccia utente grafica (GUI) dell'applicazione. Crea un file denominato `index.html`. Copia e incolla il codice seguente in. `index.html` Questo codice HTML fa riferimento`main.js`. Questa è una versione in bundle di index.js, che include i AWS SDK for JavaScript moduli richiesti.

```
<!doctype html>
<html
  xmlns:th="http://www.thymeleaf.org"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" href="./images/favicon.ico" />
    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    />
```

```
    <link rel="stylesheet" href="./css/styles.css" />
    <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
    <script src="https://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
    <script src="./js/main.js"></script>
    <style>
      .messageelement {
        margin: auto;
        border: 2px solid #dedede;
        background-color: #d7d1d0;
        border-radius: 5px;
        max-width: 800px;
        padding: 10px;
        margin: 10px 0;
      }

      .messageelement::after {
        content: "";
        clear: both;
        display: table;
      }

      .messageelement img {
        float: left;
        max-width: 60px;
        width: 100%;
        margin-right: 20px;
        border-radius: 50%;
      }

      .messageelement img.right {
        float: right;
        margin-left: 20px;
        margin-right: 0;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h2>AWS Sample Messaging Application</h2>
      <div id="messages"></div>

      <div class="input-group mb-3">
        <div class="input-group-prepend">
          <span class="input-group-text" id="basic-addon1">Sender:</span>
```

```
        </div>
        <select name="cars" id="username">
          <option value="Scott">Brian</option>
          <option value="Tricia">Tricia</option>
        </select>
      </div>

      <div class="input-group">
        <div class="input-group-prepend">
          <span class="input-group-text">Message:</span>
        </div>
        <textarea
          class="form-control"
          id="textarea"
          aria-label="With textarea"
        ></textarea>
        <button
          type="button"
          onclick="pushMessage()"
          id="send"
          class="btn btn-success"
        >
          Send
        </button>
        <button
          type="button"
          onclick="purge()"
          id="refresh"
          class="btn btn-success"
        >
          Purge
        </button>
      </div>
      <!-- All of these child items are hidden and only displayed in a FancyBox
 ------------------------------------------------------>
      <div id="hide" style="display: none">
        <div id="base" class="messageelement">
          <img
            src="../public/images/av2.png"
            alt="Avatar"
            class="right"
            style="width: 100%"
          />
          <p id="text">Excellent! So, what do you want to do today?</p>
```

```
          <span class="time-right">11:02</span>
        </div>
      </div>
    </div>
  </body>
</html>
```

Questo codice è disponibile anche qui. GitHub

Creazione dello script del browser

In questo argomento, si crea uno script del browser per l'app. Dopo aver creato lo script del browser, lo si raggruppa in un file chiamato `main.js` come descritto inRaggruppamento di JavaScript.

Crea un file denominato `index.js`. Copia e incolla il codice da qui GitHub in poi.

Questo codice è spiegato nelle seguenti sezioni:

1. Configurazione
2. Popola Chat
3. messaggi push
4. purga

Configurazione

Innanzitutto, crea una `libs` directory e crea l'oggetto client Amazon SQS richiesto creando un file denominato. `sqsClient.js` Sostituisci *`REGION`* e *`IDENTITY_POOL_ID`* in ciascuno di essi.

> **Note**
>
> Usa l'ID del pool di identità di Amazon Cognito in cui hai creato. Crea le risorse 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
  }),
});
```

In`index.js`, importa i AWS SDK for JavaScript moduli e i comandi richiesti. Sostituisci *`SQS_QUEUE_NAME con il nome`* della coda Amazon SQS che hai creato in. Crea le risorse 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.
```

Compila la chat

La `populateChat` funzione onload recupera automaticamente l'URL per la coda Amazon SQS, recupera tutti i messaggi in coda e li visualizza.

```
$(function () {
  populateChat();
});

const populateChat = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
```

```
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for retrieving the messages in the Amazon SQS Queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };
    try {
      // Retrieve the messages from the Amazon SQS Queue.
      const data = await sqsClient.send(
        new ReceiveMessageCommand(getMessageParams)
      );
      console.log("Successfully retrieved messages", data.Messages);

      // Loop through messages for user and message body.
      var i;
      for (i = 0; i < data.Messages.length; i++) {
        const name = data.Messages[i].MessageAttributes.Name.StringValue;
        const body = data.Messages[i].Body;
        // Create the HTML for the message.
        var userText = body + "<br><br><b>" + name;
        var myTextNode = $("#base").clone();
        myTextNode.text(userText);
        var image_url;
        var n = name.localeCompare("Scott");
        if (n == 0) image_url = "./images/av1.png";
        else image_url = "./images/av2.png";
        var images_div =
          '<img src="' +
          image_url +
          '" alt="Avatar" class="right" style=""width:100%;"">';
        myTextNode.html(userText);
        myTextNode.append(images_div);

        // Add the message to the GUI.
        $("#messages").append(myTextNode);
      }
    } catch (err) {
      console.log("Error loading messages: ", err);
    }
  } catch (err) {
    console.log("Error retrieving SQS queue URL: ", err);
```

```
  }
};
```

Messaggi push

L'utente seleziona il proprio nome e immette il messaggio, quindi invia il messaggio, avviando la funzione. `pushMessage` `pushMessage`recupera l'URL della coda Amazon SQS, quindi invia un messaggio con un valore ID messaggio univoco (un GUID), il testo del messaggio e l'utente alla coda Amazon SQS. Quindi recupera tutti i messaggi dalla coda di Amazon SQS e li visualizza.

```
const pushMessage = async () => {
  // Get and convert user and message input.
  var user = $("#username").val();
  var message = $("#textarea").val();

  // Create random deduplication ID.
  var dt = new Date().getTime();
  var uuid = "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (
    c
  ) {
    var r = (dt + Math.random() * 16) % 16 | 0;
    dt = Math.floor(dt / 16);
    return (c == "x" ? r : (r & 0x3) | 0x8).toString(16);
  });

  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for the message.
    var messageParams = {
      MessageAttributes: {
        Name: {
          DataType: "String",
          StringValue: user,
        },
```

```
      },
      MessageBody: message,
      MessageDeduplicationId: uuid,
      MessageGroupId: "GroupA",
      QueueUrl: data.QueueUrl,
    };
    const result = await sqsClient.send(new SendMessageCommand(messageParams));
    console.log("Success", result.MessageId);

    // Set the parameters for retrieving all messages in the SQS queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };

    // Retrieve messages from SQS Queue.
    const final = await sqsClient.send(
      new ReceiveMessageCommand(getMessageParams)
    );
    console.log("Successfully retrieved", final.Messages);
    $("#messages").empty();
    // Loop through messages for user and message body.
    var i;
    for (i = 0; i < final.Messages.length; i++) {
      const name = final.Messages[i].MessageAttributes.Name.StringValue;
      const body = final.Messages[i].Body;
      // Create the HTML for the message.
      var userText = body + "<br><br><b>" + name;
      var myTextNode = $("#base").clone();
      myTextNode.text(userText);
      var image_url;
      var n = name.localeCompare("Scott");
      if (n == 0) image_url = "./images/av1.png";
      else image_url = "./images/av2.png";
      var images_div =
        '<img src="' +
        image_url +
        '" alt="Avatar" class="right" style=""width:100%;"">';
      myTextNode.html(userText);
      myTextNode.append(images_div);
      // Add the HTML to the GUI.
```

```
      $("#messages").append(myTextNode);
    }
  } catch (err) {
    console.log("Error", err);
  }
};
// Make the function available to the browser window.
window.pushMessage = pushMessage;
```

Elimina i messaggi

`purge`elimina i messaggi da Amazon SQS Queue e dall'interfaccia utente.

```
// 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;
```

Raggruppamento di JavaScript

Questo codice completo dello script del browser è disponibile qui. GitHub .

Ora usa webpack per raggruppare i AWS SDK for JavaScript moduli `index.js` and in un unico file,. `main.js`

1. Se non l'hai già fatto, segui questo esempio Prerequisiti per installare webpack.

   > **Note**
   >
   > Per informazioni sul webpack, consulta. Raggruppa le applicazioni con webpack

2. Esegui quanto segue nella riga di comando per raggruppare il file di questo JavaScript esempio in un file chiamato: `<index.js>`

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

Passaggi successivi

Complimenti! Hai creato e distribuito l'applicazione di AWS messaggistica che utilizza Amazon SQS. Come indicato all'inizio di questo tutorial, assicurati di terminare tutte le risorse che crei durante la lettura di questo tutorial per assicurarti che non ti vengano più addebitati costi. Puoi farlo eliminando lo AWS CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

1. Apri il file AWS CloudFormationnella console di AWS gestione.
2. Apri la pagina Stacks e seleziona lo stack.
3. Scegli Delete (Elimina).

Utilizzare AWS Cloud9 con AWS SDK for JavaScript

È possibile utilizzare AWS Cloud9 con il AWS SDK for JavaScript per scrivere ed eseguire il codice JavaScript nel browser, nonché per scrivere, eseguire ed eseguire il debug del codice Node.js, utilizzando solo un browser. AWS Cloud9 include strumenti come un editor di codice e un terminale, oltre a un debugger per il codice Node.js.

Poiché l' AWS Cloud9 IDE è basato sul cloud, puoi lavorare sui tuoi progetti dall'ufficio, da casa o ovunque utilizzando una macchina connessa a Internet. Per informazioni generali su AWS Cloud9, consulta la Guida per l'AWS Cloud9 utente.

I passaggi seguenti descrivono come configurare AWS Cloud9 con l'SDK per JavaScript.

Indice

Passaggio 1: configura il tuo AWS account da utilizzare AWS Cloud9

Inizia a utilizzarlo AWS Cloud9 accedendo alla AWS Cloud9 console come entità AWS Identity and Access Management (IAM) (ad esempio, un utente IAM) con le autorizzazioni di accesso per AWS Cloud9 il tuo AWS account.

Per configurare un'entità IAM nel tuo AWS account per l'accesso e per accedere alla AWS Cloud9 console AWS Cloud9, consulta la sezione Configurazione del team AWS Cloud9 nella Guida per l'AWS Cloud9 utente.

Fase 2: configura il tuo ambiente di AWS Cloud9 sviluppo

Dopo aver effettuato l'accesso alla AWS Cloud9 console, utilizza la console per creare un ambiente di AWS Cloud9 sviluppo. Dopo aver creato l'ambiente, AWS Cloud9 apre l'IDE per quell'ambiente.

Per ulteriori informazioni, consulta Creazione di un ambiente AWS Cloud9 nella Guida per AWS Cloud9 l'utente.

> **Note**
>
> Dal momento che stai creando l'ambiente nella console per la prima volta, ti consigliamo di scegliere l'opzione Create a new instance for environment (EC2) (Crea una nuova istanza per l'ambiente (EC2)). Questa opzione indica AWS Cloud9 di creare un ambiente, avviare un'istanza Amazon EC2 e quindi connettere la nuova istanza al nuovo ambiente. Questo è il modo più veloce per iniziare a usare AWS Cloud9.

Passaggio 3: configura l'SDK per JavaScript

Dopo aver AWS Cloud9 aperto l'IDE per il tuo ambiente di sviluppo, segui una o entrambe le seguenti procedure per utilizzare l'IDE per configurare l'SDK JavaScript nel tuo ambiente.

Per configurare l'SDK per Node.js JavaScript

1. Apri il terminale se non è già aperto nell'IDE. Per eseguire questa operazione, nella barra del menu nell'IDE, scegli Window, New Terminal (Finestra, Nuovo terminale).
2. Esegui il comando seguente da utilizzare `npm` per installare il `Cloud9` client dell'SDK per. JavaScript

   ```
   npm install @aws-sdk/client-cloud9
   ```

   Se l'IDE non riesce a trovarlo`npm`, esegui i seguenti comandi, uno alla volta nell'ordine seguente, per l'installazione`npm`. (Questi comandi prevedono che tu abbia scelto, nella fase precedente, l'opzione Create a new instance for environment (EC2) (Crea una nuova istanza per l'ambiente (EC2))).

> **Warning**
>
> AWS non controlla il codice seguente. Prima di eseguirlo, assicurati di verificarne l'autenticità e l'integrità. Ulteriori informazioni su questo codice sono disponibili nel repository nvm (Node Version Manager) GitHub .

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash #
 Download and install Node Version Manager (nvm).
. ~/.bashrc                                                                    #
 Activate nvm.
nvm install node                                                               #
 Use nvm to install npm (and Node.js at the same time).
```

Per configurare l'SDK per nel browser JavaScript

Per utilizzare l'SDK JavaScript nelle pagine HTML, utilizzatelo WebPack per raggruppare i moduli client richiesti e tutte le JavaScript funzioni richieste in un unico JavaScript file e aggiungetelo in un tag script nelle <head> pagine HTML. Per esempio:

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

> **Note**
>
> Per ulteriori informazioni su Webpack, consulta Raggruppa le applicazioni con webpack

Passaggio 4: scarica il codice di esempio

Usa il terminale che hai aperto nel passaggio precedente per scaricare il codice di esempio per l'SDK JavaScript nell'ambiente di AWS Cloud9 sviluppo. (Se il terminale non è già aperto nell'IDE, aprilo scegliendo Window, New Terminal (Finestra, Nuovo Terminal) sulla barra dei menu nell'IDE).

Esegui il seguente comando per scaricare il codice di esempio. Questo comando scarica una copia di tutti gli esempi di codice utilizzati nella documentazione ufficiale dell' AWS SDK nella directory principale dell'ambiente.

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

Per trovare esempi di codice per l'SDK for JavaScript, utilizzate la finestra Ambiente per aprire*ENVIRONMENT_NAME*\aws-doc-sdk-examples\javascriptv3\example_code/src, dove *ENVIRONMENT_NAME è il nome del vostro ambiente* di sviluppo. AWS Cloud9

Per imparare a lavorare con questi e altri esempi di codice, consulta SDK per esempi di codice. JavaScript

Passaggio 5: Esegui ed esegui il debug del codice di esempio

Per eseguire il codice nel tuo ambiente di AWS Cloud9 sviluppo, consulta Esegui il codice nella Guida per l'AWS Cloud9 utente.

Per eseguire il debug del codice Node.js, consulta Eseguire il debug del codice nella Guida per l'AWS Cloud9 utente.

SDK per esempi di JavaScript codice (v3)

Gli esempi di codice in questo argomento mostrano come utilizzare AWS SDK for JavaScript (v3) con. AWS

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Esempi cross-service: applicazioni di esempio che funzionano su più servizi Servizi AWS.

Esempi

- Azioni e scenari che utilizzano SDK for JavaScript (v3)
- Esempi interservizi che utilizzano SDK for JavaScript (v3)

Azioni e scenari che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. Servizi AWS

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Servizi

- Esempi di Auto Scaling con SDK for JavaScript (v3)
- Esempi di Amazon Bedrock con SDK for JavaScript (v3)
- Esempi di Amazon Bedrock Runtime con SDK for JavaScript (v3)
- Esempi di Agents for Amazon Bedrock che utilizzano SDK for JavaScript (v3)

- Esempi di Agents for Amazon Bedrock Runtime con SDK for JavaScript (v3)
- CloudWatch esempi che utilizzano SDK for JavaScript (v3)
- CloudWatch Esempi di eventi che utilizzano SDK for JavaScript (v3)
- CloudWatch Registra esempi utilizzando SDK for JavaScript (v3)
- CodeBuild esempi che utilizzano SDK for JavaScript (v3)
- Esempi di Amazon Cognito Identity Provider che utilizzano SDK for JavaScript (v3)
- Esempi di DynamoDB che utilizzano SDK JavaScript for (v3)
- Esempi di Amazon EC2 con SDK for JavaScript (v3)
- Esempi di Elastic Load Balancing con SDK for JavaScript (v3)
- EventBridge esempi che utilizzano SDK for JavaScript (v3)
- AWS Glue esempi che utilizzano SDK for JavaScript (v3)
- HealthImaging esempi che utilizzano SDK for JavaScript (v3)
- Esempi IAM che utilizzano SDK for JavaScript (v3)
- Esempi di Lambda con SDK for JavaScript (v3)
- Esempi di Amazon Personalize utilizzando SDK for JavaScript (v3)
- Esempi di Amazon Personalize Events utilizzando SDK for JavaScript (v3)
- Esempi di Amazon Personalize Runtime utilizzando SDK for JavaScript (v3)
- Esempi di Amazon Pinpoint con SDK for JavaScript (v3)
- Esempi di Amazon Redshift con SDK for JavaScript (v3)
- Esempi di Amazon S3 con SDK for JavaScript (v3)
- Esempi di S3 Glacier che utilizzano SDK for (v3) JavaScript
- SageMaker esempi che utilizzano SDK for JavaScript (v3)
- Esempi di Secrets Manager con SDK for JavaScript (v3)
- Esempi di Amazon SES con SDK for JavaScript (v3)
- Esempi di Amazon SNS con SDK for JavaScript (v3)
- Esempi di Amazon SQS con SDK for JavaScript (v3)
- Esempi di Step Functions con SDK for JavaScript (v3)
- AWS STS esempi che utilizzano SDK for JavaScript (v3)
- AWS Support esempi che utilizzano SDK for JavaScript (v3)

- Esempi di Amazon Transcribe con SDK JavaScript for (v3)

Esempi di Auto Scaling con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Auto Scaling.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni
- Scenari

Azioni

Associare un gruppo target ELB a un gruppo Auto Scaling

Il seguente esempio di codice mostra come collegare un gruppo target ELB a un gruppo Auto Scaling.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const client = new AutoScalingClient({});
await client.send(
```

```
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
```

- Per i dettagli sull'API, consulta la AttachLoadBalancerTargetGroupssezione AWS SDK for JavaScript API Reference.

Scenari

Creazione e gestione di un servizio resiliente

Il seguente esempio di codice mostra come creare un servizio Web con bilanciamento del carico che restituisca consigli su libri, film e canzoni. L'esempio mostra come il servizio risponde ai guasti e spiega come ristrutturarlo per una maggiore resilienza in caso di guasti.

- Utilizza un gruppo con dimensionamento automatico Amazon EC2 per creare istanze Amazon Elastic Compute Cloud (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze entro un intervallo specificato.
- Gestisci e distribuisci le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo con dimensionamento automatico e inoltra le richieste soltanto alle istanze integre.
- Esegui un server Web Python su ogni istanza EC2 per gestire le richieste HTTP. Il server Web risponde con consigli e controlli dell'integrità.
- Simula un servizio di raccomandazione con una tabella Amazon DynamoDB.
- Controlla la risposta del server web alle richieste e ai controlli sanitari aggiornando AWS Systems Manager i parametri.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui lo scenario interattivo al prompt dei comandi.

```
#!/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);
}
```

Crea passaggi per distribuire tutte le risorse.

```
// 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,
      );
    }
  }),
];
```

Crea i passaggi per eseguire la demo.

```
// 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: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      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;
}
```

Crea i passaggi per distruggere tutte le risorse.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Esempi di Amazon Bedrock con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Bedrock.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve Amazon Bedrock

I seguenti esempi di codice mostrano come iniziare a usare Amazon Bedrock.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
}
```

- Per i dettagli sull'API, consulta la ListFoundationModelssezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni

Azioni

Ottieni dettagli su un modello di base Amazon Bedrock

Il seguente esempio di codice mostra come ottenere dettagli su un modello di base Amazon Bedrock.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottieni dettagli su un modello di base.

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

import { fileURLToPath } from "url";

import {
  BedrockClient,
  GetFoundationModelCommand,
} from "@aws-sdk/client-bedrock";

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 models.
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();
```

```
  const command = new GetFoundationModelCommand({
    modelIdentifier: "amazon.titan-embed-text-v1",
  });

  const response = await client.send(command);

  return response.modelDetails;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const model = await getFoundationModel();
  console.log(model);
}
```

- Per i dettagli sull'API, consulta la GetFoundationModelsezione AWS SDK for JavaScript API Reference.

Elenca i modelli Amazon Bedrock Foundation disponibili

Il seguente esempio di codice mostra come elencare i modelli Amazon Bedrock Foundation disponibili.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i modelli di base disponibili.

```
// 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);
}
```

- Per i dettagli sulle API, consulta la ListFoundationModelssezione AWS SDK for JavaScript API Reference.

Esempi di Amazon Bedrock Runtime con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Bedrock Runtime.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

Generazione di testo con AI21 Labs Jurassic-2

Il seguente esempio di codice mostra come richiamare il modello AI21 Labs Jurassic-2 su Amazon Bedrock per la generazione di testo.

JavaScript SDK per (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invoca il modello di base Jurassic-2 di AI21 Labs per generare testo.

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

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  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 the AI21 Labs Jurassic-2 large-language model to run an inference
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Jurassic-2 to complete.
 * @returns {string} The inference response (completion) from the model.
 */
export const invokeJurassic2 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "ai21.j2-mid-v1";

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for AI21 Labs Jurassic-2, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
jurassic2.html
   */
  const payload = {
    prompt,
    maxTokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);
```

```
    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.completions[0].data.text;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: AI21 Labs Jurassic-2");
  console.log(`Prompt: ${prompt}`);

  const completion = await invokeJurassic2(prompt);
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- Per i dettagli sulle API, consulta la sezione API Reference. InvokeModelAWS SDK for JavaScript

Generazione di testo con Amazon Titan Text G1

Il seguente esempio di codice mostra come richiamare il modello Amazon Titan Text G1 su Amazon Bedrock per la generazione di testo.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Richiama il modello di base Amazon Titan Text G1 per generare testo.

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

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {Object[]} results
 */

/**
 * Invokes the Titan Text G1 - Express model to run an inference
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Titan Text Express to complete.
 * @returns {object[]} The inference response (results) from the model.
 */
export const invokeTitanTextExpressV1 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "amazon.titan-text-express-v1";

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Titan text, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-titan-
text.html
   */
```

```
  const textGenerationConfig = {
    maxTokenCount: 4096,
    stopSequences: [],
    temperature: 0,
    topP: 1,
  };

  const payload = {
    inputText: prompt,
    textGenerationConfig,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);
    return responseBody.results;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = `Meeting transcript: Miguel: Hi Brant, I want to discuss the
 workstream
    for our new product launch Brant: Sure Miguel, is there anything in particular
 you want
```

```
    to discuss? Miguel: Yes, I want to talk about how users enter into the product.
    Brant: Ok, in that case let me add in Namita. Namita: Hey everyone
    Brant: Hi Namita, Miguel wants to discuss how users enter into the product.
    Miguel: its too complicated and we should remove friction.
    for example, why do I need to fill out additional forms?
    I also find it difficult to find where to access the product
    when I first land on the landing page. Brant: I would also add that
    I think there are too many steps. Namita: Ok, I can work on the
    landing page to make the product more discoverable but brant
    can you work on the additonal forms? Brant: Yes but I would need
    to work with James from another team as he needs to unblock the sign up
 workflow.
    Miguel can you document any other concerns so that I can discuss with James only
 once?
    Miguel: Sure.
    From the meeting transcript above, Create a list of action items for each
 person.`;

  console.log("\nModel: Titan Text Express v1");
  console.log(`Prompt: ${prompt}`);

  const results = await invokeTitanTextExpressV1(prompt);
  console.log("Completion:");
  for (const result of results) {
    console.log(result.outputText);
  }
  console.log("\n");
}
```

- Per i dettagli sull'API, consulta la sezione API InvokeModelReference AWS SDK for JavaScript .

Generazione di testo con Anthropic Claude 2

Il seguente esempio di codice mostra come richiamare il modello Anthropic Claude 2 su Amazon Bedrock per la generazione di testo.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invoca il modello di base Anthropic Claude 2 per generare testo.

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

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes the Anthropic Claude 2 model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Claude to complete.
 * @returns {string} The inference response (completion) from the model.
 */
export const invokeClaude = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "anthropic.claude-v2";

  /* Claude requires you to enclose the prompt as follows: */
  const enclosedPrompt = `Human: ${prompt}\n\nAssistant:`;

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Anthropic Claude, refer to:
```

```
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
claude.html
   */
  const payload = {
    prompt: enclosedPrompt,
    max_tokens_to_sample: 500,
    temperature: 0.5,
    stop_sequences: ["\n\nHuman:"],
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.completion;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Anthropic Claude v2");
  console.log(`Prompt: ${prompt}`);

  const completion = await invokeClaude(prompt);
```

```
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- Per i dettagli sulle API, consulta la sezione API InvokeModelReference AWS SDK for JavaScript .

Generazione di testo con Meta Llama 2 Chat

Il seguente esempio di codice mostra come richiamare il modello Meta Llama 2 Chat su Amazon Bedrock per la generazione di testo.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invoca il modello base di Meta Llama 2 Chat per generare testo.

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

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {generation} text
 */

/**
 * Invokes the Meta Llama 2 Chat model to run an inference
```

```
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Llama-2 to complete.
 * @returns {string} The inference response (generation) from the model.
 */
export const invokeLlama2 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "meta.llama2-13b-chat-v1";

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Meta Llama 2 Chat, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-meta.html
   */
  const payload = {
    prompt,
    temperature: 0.5,
    top_p: 0.9,
    max_gen_len: 512,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.generation;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
```

```
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Meta Llama 2 Chat");
  console.log(`Prompt: ${prompt}`);

  const completion = await invokeLlama2(prompt);
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API InvokeModelReference.

Generazione di testo con Mistral 7B

Il seguente esempio di codice mostra come richiamare il modello Mistral 7B su Amazon Bedrock per la generazione di testo.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invoca il modello di base Mistral 7B per generare testo.

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

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
```

```
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes the Mistral 7B model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Mistral to complete.
 * @returns {string[]} A list of inference responses (completions) from the model.
 */
export const invokeMistral7B = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-west-2" });

  const modelId = "mistral.mistral-7b-instruct-v0:2";

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `<s>[INST] ${prompt} [/INST]`;

  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);
```

```
    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.outputs.map((output) => output.text);
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Mistral 7B");
  console.log(`Prompt: ${prompt}`);

  const completions = await invokeMistral7B(prompt);
  completions.forEach((completion) => {
    console.log("Completion:");
    console.log(completion);
    console.log("\n");
  });
}
```

- Per i dettagli sull'API, vedere InvokeModelin API Reference.AWS SDK for JavaScript

Generazione di testo con Mixtral 8x7B

Il seguente esempio di codice mostra come richiamare il modello di modello Mixtral 8x7B su Amazon Bedrock per la generazione di testo.

JavaScript SDK per (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invoca il modello di base Mixtral 8x7B per generare testo.

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

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
import { invokeMistral7B } from "./invoke-mistral7b.js";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes the Mixtral 8x7B model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Mistral to complete.
 * @returns {string[]} A list of inference responses (completions) from the model.
 */
export const invokeMixtral8x7B = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-west-2" });

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `<s>[INST] ${prompt} [/INST]`;
```

```
  const modelId = "mistral.mixtral-8x7b-instruct-v0:1";

  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.outputs.map((output) => output.text);
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Mixtral 8x7B");
  console.log(`Prompt: ${prompt}`);

  const completions = await invokeMistral7B(prompt);
  completions.forEach((completion) => {
```

```
    console.log("Completion:");
    console.log(completion);
    console.log("\n");
  });
}
```

- Per i dettagli sulle API, consultate la sezione API Reference. InvokeModelAWS SDK for JavaScript

Esempi di Agents for Amazon Bedrock che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with Agents for Amazon Bedrock.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Agents per Amazon Bedrock

Il seguente esempio di codice mostra come iniziare a usare Agents for Amazon Bedrock.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - GetAgent
  - ListAgents

Argomenti

- Azioni

Azioni

Creazione di un agente

Il seguente esempio di codice mostra come creare un agente Amazon Bedrock.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di 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,
  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);
}
```

- Per i dettagli sull'API, consulta la CreateAgentsezione AWS SDK for JavaScript API Reference.

Eliminazione di un agente

Il seguente esempio di codice mostra come eliminare un agente Amazon Bedrock.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare 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);
}
```

- Per i dettagli sull'API, consulta la DeleteAgentsezione AWS SDK for JavaScript API Reference.

Come ottenere informazioni su un agente

Il seguente esempio di codice mostra come ottenere informazioni su un agente Amazon Bedrock.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Trovate 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);
```

```
}
```

- Per i dettagli sull'API, GetAgentconsulta AWS SDK for JavaScript API Reference.

Elenca i gruppi di azione per un agente

Il seguente esempio di codice mostra come elencare i gruppi di azioni per un agente Amazon Bedrock.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i gruppi d'azione di 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);
  }
}
```

- Per i dettagli sull'API, vedere ListAgentActionGroupsin AWS SDK for JavaScript API Reference.

Elenca gli agenti disponibili

Il seguente esempio di codice mostra come elencare gli Agents for Amazon Bedrock appartenenti a un account.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca gli agenti appartenenti a un account.

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

- Per i dettagli sull'API, consulta la ListAgentssezione AWS SDK for JavaScript API Reference.

Esempi di Agents for Amazon Bedrock Runtime con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with Agents for Amazon Bedrock Runtime.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Invoca un agente

Il seguente esempio di codice mostra come richiamare un agente Amazon Bedrock.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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);
}
```

- Per i dettagli sull'API, consulta la InvokeAgentsezione AWS SDK for JavaScript API Reference.

CloudWatch esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. CloudWatch

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Creazione un allarme per un parametro

Il seguente esempio di codice mostra come creare o aggiornare un CloudWatch allarme Amazon e associarlo alla metrica, all'espressione matematica della metrica, al modello di rilevamento delle anomalie o alla query Metrics Insights specificati.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutMetricAlarmsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutMetricAlarmsezione AWS SDK for JavaScript API Reference.

Eliminare allarmi

Il seguente esempio di codice mostra come eliminare gli CloudWatch allarmi Amazon.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteAlarmssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteAlarmssezione AWS SDK for JavaScript API Reference.

Descrivi allarmi per un parametro

Il seguente esempio di codice mostra come descrivere gli CloudWatch allarmi Amazon per una metrica.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DescribeAlarmsForMetricsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DescribeAlarmsForMetricsezione AWS SDK for JavaScript API Reference.

Disattivare le operazioni di allarme

Il seguente esempio di codice mostra come disabilitare le azioni di CloudWatch allarme di Amazon.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DisableAlarmActionssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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);
    }
  }
);
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DisableAlarmActionssezione AWS SDK for JavaScript API Reference.

Attivare le operazioni di allarme

Il seguente esempio di codice mostra come abilitare le azioni di CloudWatch allarme di Amazon.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la EnableAlarmActionssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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);
      }
    });
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la EnableAlarmActionssezione AWS SDK for JavaScript API Reference.

Elencare parametri

Il seguente esempio di codice mostra come elencare i metadati per le CloudWatch metriche di Amazon. Per ottenere i dati per una metrica, usa le GetMetricData azioni o. GetMetricStatistics

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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);
};
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListMetricssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListMetricssezione AWS SDK for JavaScript API Reference.

Inserimento dei dati in un parametro

Il seguente esempio di codice mostra come pubblicare punti dati metrici su Amazon CloudWatch.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutMetricDatasezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutMetricDatasezione AWS SDK for JavaScript API Reference.

CloudWatch Esempi di eventi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with CloudWatch Events.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Aggiunta di un tag come destinazione

Il seguente esempio di codice mostra come aggiungere un target a un evento Amazon CloudWatch Events.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutTargetssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutTargetssezione AWS SDK for JavaScript API Reference.

Creazione di una regola pianificata

Il seguente esempio di codice mostra come creare una regola pianificata di Amazon CloudWatch Events.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutRulesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutRulesezione AWS SDK for JavaScript API Reference.

Invio di eventi

Il seguente esempio di codice mostra come inviare CloudWatch eventi Amazon Events.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutEventssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutEventssezione AWS SDK for JavaScript API Reference.

CloudWatch Registra esempi utilizzando SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with Logs. CloudWatch

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni
- Scenari

Azioni

Creazione di un gruppo di log

Il seguente esempio di codice mostra come creare un nuovo gruppo di log CloudWatch Logs.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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();
```

- Per i dettagli sull'API, consulta la CreateLogGroupsezione AWS SDK for JavaScript API Reference.

Creazione di un filtro di sottoscrizione

Il seguente esempio di codice mostra come creare un filtro di abbonamento Amazon CloudWatch Logs.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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();
```

- Per i dettagli sull'API, consulta la PutSubscriptionFiltersezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutSubscriptionFiltersezione AWS SDK for JavaScript API Reference.

Eliminazione di un gruppo di log

Il seguente esempio di codice mostra come eliminare un gruppo di log CloudWatch Logs esistente.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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();
```

- Per i dettagli sull'API, consulta la DeleteLogGroupsezione AWS SDK for JavaScript API Reference.

Eliminazione di un filtro di sottoscrizione

Il seguente esempio di codice mostra come eliminare un filtro di abbonamento Amazon CloudWatch Logs.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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();
```

- Per i dettagli sull'API, consulta la DeleteSubscriptionFiltersezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteSubscriptionFiltersezione AWS SDK for JavaScript API Reference.

Descrizione dei filtri di sottoscrizione esistenti

Il seguente esempio di codice mostra come descrivere i filtri di abbonamento esistenti di Amazon CloudWatch Logs.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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();
```

- Per i dettagli sull'API, consulta la DescribeSubscriptionFilterssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DescribeSubscriptionFilterssezione AWS SDK for JavaScript API Reference.

Descrizione di gruppi di log

Il seguente esempio di codice mostra come descrivere i gruppi di log di CloudWatch Logs.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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;
};
```

- Per i dettagli sull'API, consulta la DescribeLogGroupssezione AWS SDK for JavaScript API Reference.

Ottieni i risultati di una query

Il seguente esempio di codice mostra come ottenere i risultati di una query.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
```

- Per i dettagli sull'API, consulta la GetQueryResultssezione AWS SDK for JavaScript API Reference.

Avvio di una sessione Live Tail

Il seguente esempio di codice mostra come avviare una sessione Live Tail per un gruppo/flusso di log esistente.

SDK per (v3 JavaScript)

Includere i file richiesti.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-
cloudwatch-logs";
```

Gestisci gli eventi della sessione di 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)
    }
}
```

Avvia la sessione 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);
    }
```

Interrompi la sessione Live Tail dopo un certo periodo di tempo.

```
    /* Set a timeout to close the client. This will stop the Live Tail session. */
    setTimeout(function() {
        console.log("Client timeout");
        client.destroy();
      }, 10000);
```

- Per i dettagli sulle API, consulta la sezione AWS SDK for JavaScript API StartLiveTailReference.

Avvio di una query

Il seguente esempio di codice mostra come avviare una query.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
/**
 * 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;
    }
  }
```

- Per i dettagli sull'API, consulta la StartQuerysezione AWS SDK for JavaScript API Reference.

Scenari

Esegui una query di grandi dimensioni

Il seguente esempio di codice mostra come utilizzare CloudWatch Logs per interrogare più di 10.000 record.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo è il punto di ingresso.

```
// 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}`,
);
```

Questa è una classe che divide le interrogazioni in più fasi, se necessario.

```
// 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;
      },
    );
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - GetQueryResults
  - StartQuery

CodeBuild esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. CodeBuild

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Crea un progetto

Il seguente esempio di codice mostra come creare un CodeBuild progetto.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un progetto.

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
} from "@aws-sdk/client-codebuild";

// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});

  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
        location: buildOutputBucket,
```

```
      },
      // Information about the build environment. The combination of "computeType"
 and "type" determines the
      // requirements for the environment such as CPU, memory, and disk space.
      environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
compute-types.html
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
      },
      name: projectName,
      // A role ARN with permission to create a CodeBuild project, write to the
 artifact location, and write CloudWatch logs.
      serviceRole: roleArn,
      source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be built.
        location: githubUrl,
      },
    }),
  );
  console.log(response);
  //   {
  //     '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //     },
  //     project: {
  //       arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
  //       artifacts: {
  //         encryptionDisabled: false,
  //         location: 'xxxxxx-xxxxxxx-xxxxxx',
  //         name: 'MyCodeBuilder',
```

```
  //         namespaceType: 'NONE',
  //         packaging: 'NONE',
  //         type: 'S3'
  //       },
  //       badge: { badgeEnabled: false },
  //       cache: { type: 'NO_CACHE' },
  //       created: 2023-08-18T14:46:48.979Z,
  //       encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
  //       environment: {
  //         computeType: 'BUILD_GENERAL1_SMALL',
  //         environmentVariables: [],
  //         image: 'aws/codebuild/standard:7.0',
  //         imagePullCredentialsType: 'CODEBUILD',
  //         privilegedMode: false,
  //         type: 'LINUX_CONTAINER'
  //       },
  //       lastModified: 2023-08-18T14:46:48.979Z,
  //       name: 'MyCodeBuilder',
  //       projectVisibility: 'PRIVATE',
  //       queuedTimeoutInMinutes: 480,
  //       serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
  //       source: {
  //         insecureSsl: false,
  //         location: 'https://...',
  //         reportBuildStatus: false,
  //         type: 'GITHUB'
  //       },
  //       timeoutInMinutes: 60
  //     }
  //   }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la CreateProjectsezione AWS SDK for JavaScript API Reference.

Esempi di Amazon Cognito Identity Provider che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Cognito Identity Provider.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Ciao Amazon Cognito

Gli esempi di codice seguente mostrano come iniziare a utilizzare Amazon Cognito.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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;
};
```

- Per i dettagli sull'API, consulta la ListUserPoolssezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Conferma di un utente

Il seguente esempio di codice mostra come confermare un utente Amazon Cognito.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
```

```
    ConfirmationCode: code,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la ConfirmSignUpsezione AWS SDK for JavaScript API Reference.

Verifica del monitoraggio di un dispositivo MFA

Il seguente esempio di codice mostra come confermare un dispositivo MFA per il tracciamento da parte di Amazon Cognito.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
};
```

- Per i dettagli sull'API, consulta la ConfirmDevicesezione AWS SDK for JavaScript API Reference.

Recupero di un token per associare un'applicazione MFA a un utente

Il seguente esempio di codice mostra come ottenere un token per associare un'applicazione MFA a un utente Amazon Cognito.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la AssociateSoftwareTokensezione AWS SDK for JavaScript API Reference.

Recupero delle informazioni su un utente

Il seguente esempio di codice mostra come ottenere informazioni su un utente di Amazon Cognito.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const adminGetUser = ({ userPoolId, username }) => {
```

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

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
    Username: username,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la AdminGetUsersezione AWS SDK for JavaScript API Reference.

Elencare gli utenti

Il seguente esempio di codice mostra come elencare gli utenti di Amazon Cognito.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la ListUserssezione AWS SDK for JavaScript API Reference.

Rinvio di un codice di conferma

Il seguente esempio di codice mostra come inviare nuovamente un codice di conferma di Amazon Cognito.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la ResendConfirmationCodesezione AWS SDK for JavaScript API Reference.

Risposta alle richieste di autenticazione SRP

Il seguente esempio di codice mostra come rispondere alle sfide di autenticazione SRP di Amazon Cognito.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
};
```

- Per i dettagli sull'API, consulta la RespondToAuthChallengesezione AWS SDK for JavaScript API Reference.

Risposta a una richiesta di autenticazione

Il seguente esempio di codice mostra come rispondere a una sfida di autenticazione di Amazon Cognito.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const 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);
};
```

- Per i dettagli sull'API, consulta la AdminRespondToAuthChallengesezione AWS SDK for JavaScript API Reference.

Registrazione di un utente

Il seguente esempio di codice mostra come registrare un utente con Amazon Cognito.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const 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);
};
```

- Per i dettagli sull'API, consulta la SignUpsezione AWS SDK for JavaScript API Reference.

Avvio dell'autenticazione

Il seguente esempio di codice mostra come avviare l'autenticazione con Amazon Cognito.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const 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);
};
```

- Per i dettagli sull'API, consulta la InitiateAuthsezione AWS SDK for JavaScript API Reference.

Avvio dell'autenticazione con le credenziali di amministratore

Il seguente esempio di codice mostra come avviare l'autenticazione con Amazon Cognito e le credenziali di amministratore.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
};
```

- Per i dettagli sull'API, consulta la AdminInitiateAuthsezione AWS SDK for JavaScript API Reference.

Verifica di un'applicazione MFA con un utente

Il seguente esempio di codice mostra come verificare un'applicazione MFA con un utente Amazon Cognito.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
};
```

- Per i dettagli sull'API, consulta la VerifySoftwareTokensezione AWS SDK for JavaScript API Reference.

Scenari

Registrazione di un utente a un pool di utenti che richiede l'autenticazione MFA

L'esempio di codice seguente mostra come:

- Registra e conferma un utente con nome utente, password e indirizzo e-mail.
- Configura l'autenticazione a più fattori associando un'applicazione MFA all'utente.

- Accedi utilizzando una password e un codice MFA.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per un'esperienza ottimale, clona il GitHub repository ed esegui questo esempio. Il codice seguente rappresenta un esempio dell'applicazione completa.

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up'
 command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
```

```
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Signing up.`);
    await signUp({ clientId, username, password, email });
    log(`Signed up. A confirmation email has been sent to: ${email}.`);
    log(`Run 'confirm-sign-up ${username} <code>' to confirm your account.`);
  } catch (err) {
    log(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
```

```
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-
sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Confirming user.`);
    await confirmSignUp({ clientId, username, code });
    log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
    );
  } catch (err) {
    log(err);
  }
};

export { confirmSignUpHandler };

const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});
```

```
  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../../actions/associate-software-
token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-
token' to finish the setup.",
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
```

```
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-initiate-
auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);

    log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
```

```
      log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    log(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../actions/admin-respond-to-auth-
challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-
respond-to-auth-challenge' command.`,
    );
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an
 argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
```

```
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    log("Successfully authenticated.");
  } catch (err) {
    log(err);
  }
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
```

```
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-
software-token' command.`,
    );
  }
};
const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;

  try {
    validateTotp(totp);

    log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    console.log(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
```

```
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

Esempi di DynamoDB che utilizzano SDK JavaScript for (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con DynamoDB.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello DynamoDB

Gli esempi di codice seguenti mostrano come iniziare a utilizzare DynamoDB.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
};
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni

- Scenari

Azioni

Creare una tabella

Il seguente esempio di codice mostra come creare una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la CreateTablesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la CreateTablesezione AWS SDK for JavaScript API Reference.

Eliminazione di una tabella

Il seguente esempio di codice mostra come eliminare una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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;
};
```

- Per i dettagli sull'API, consulta la DeleteTablesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteTablesezione AWS SDK for JavaScript API Reference.

Elimina una voce da una tabella

Il seguente esempio di codice mostra come eliminare un elemento da una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sulle API, consulta la DeleteItemsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina un item dalla tabella.

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

Elimina un elemento da una tabella utilizzando il client documento 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteItemsezione AWS SDK for JavaScript API Reference.

Ottenimento di un batch di elementi

Il seguente esempio di codice mostra come ottenere un batch di elementi DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sulle API, consulta la BatchGetItemsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per i dettagli sull'API, consulta la BatchGetItemsezione AWS SDK for JavaScript API Reference.

Ottieni un elemento da una tabella

Il seguente esempio di codice mostra come ottenere un elemento da una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la GetItemsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottieni un elemento da una tabella.

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

Ottieni un elemento da una tabella utilizzando il client documento 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetItemsezione AWS SDK for JavaScript API Reference.

Ottieni informazioni su una tabella

Il seguente esempio di codice mostra come ottenere informazioni su una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DescribeTablesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DescribeTablesezione AWS SDK for JavaScript API Reference.

Elencare tabelle

Il seguente esempio di codice mostra come elencare le tabelle DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK for JavaScript API Reference.

Inserisci un elemento in una tabella

Il seguente esempio di codice mostra come inserire un elemento in una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la PutItemsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inserisci un elemento in una tabella.

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

Inserisci un elemento in una tabella utilizzando il client documento 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutItemsezione AWS SDK for JavaScript API Reference.

Esecuzione di una query su una tabella

Il seguente esempio di codice mostra come eseguire una query su una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for JavaScript .

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for JavaScript .

Esecuzione di un'istruzione PartiQL

Il seguente esempio di codice mostra come eseguire un'istruzione PartiQL su una tabella DynamoDB.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di un elemento mediante 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;
};
```

Ottenimento di un elemento mediante 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;
};
```

Aggiornamento di un elemento mediante 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;
};
```

Eliminazione di un elemento mediante 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;
};
```

- Per i dettagli sull'API, consulta la ExecuteStatementsezione AWS SDK for JavaScript API Reference.

Esecuzione di batch di istruzioni PartiQL

Il seguente esempio di codice mostra come eseguire batch di istruzioni PartiQL su una tabella DynamoDB.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di un batch di elementi mediante 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;
};
```

Ottenimento di un batch di elementi mediante 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;
};
```

Aggiornamento di un batch di elementi mediante 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;
```

```
};
```

Eliminazione di un batch di elementi mediante 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;
};
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK for JavaScript API Reference.

Esegui la scansione di una tabella

Il seguente esempio di codice mostra come eseguire la scansione di una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK for JavaScript .

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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 + ")"
      );
    });
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK for JavaScript .

Aggiorna un elemento in una tabella

Il seguente esempio di codice mostra come aggiornare un elemento in una tabella DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la UpdateItemsezione AWS SDK for JavaScript API Reference.

Scrittura di un batch di elementi

Il seguente esempio di codice mostra come scrivere un batch di elementi DynamoDB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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);
  }
};
```

- Per i dettagli sulle API, consulta la BatchWriteItemsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la BatchWriteItemsezione AWS SDK for JavaScript API Reference.

Scenari

Nozioni di base sull'utilizzo di tabelle, elementi e query

L'esempio di codice seguente mostra come:

- Crea una tabella in grado di contenere i dati del filmato.
- Inserisci, ottieni e aggiorna un singolo filmato nella tabella.
- Scrivi i dati del filmato nella tabella da un file JSON di esempio.
- Esegui una query sui filmati che sono stati rilasciati in un dato anno.
- Cerca i filmati che sono stati distribuiti in diversi anni.
- Elimina un filmato dalla tabella, quindi elimina la tabella.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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.");
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan

- UpdateItem

Esecuzione di una query su una tabella mediante batch di istruzioni PartiQL

L'esempio di codice seguente mostra come:

- Ricezione di un batch di elementi mediante più istruzioni SELECT.
- Aggiunta di un batch di articoli eseguendo più istruzioni INSERT.
- Aggiornamento di un batch di elementi mediante più istruzioni UPDATE.
- Eliminazione di un batch di elementi mediante più istruzioni DELETE.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eseguire le istruzioni PartiQL in batch.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async () => {
```

```
  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
```

```
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log(`Cities inserted.`);

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
```

```
  );

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log(`Updated cities.`);

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  await docClient.send(deleteItemStatementCommand);
```

```
  log("Cities deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK for JavaScript API Reference.

Esecuzione di una query mediante PartiQL

L'esempio di codice seguente mostra come:

- Ricezione di un articolo eseguendo un'istruzione SELECT.
- Aggiunta di un elemento eseguendo un'istruzione INSERT.
- Aggiornamento di un elemento eseguendo un'istruzione UPDATE.
- Eliminazione di un elemento eseguendo un'istruzione DELETE.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eseguire le singole istruzioni PartiQL.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
```

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

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

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async () => {
  /**
   * 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.");
};
```

- Per i dettagli sull'API, consulta la ExecuteStatementsezione AWS SDK for JavaScript API Reference.

Esempi di Amazon EC2 con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon EC2.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Amazon EC2

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la DescribeSecurityGroupssezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Allocare un indirizzo IP elastico

Il seguente esempio di codice mostra come allocare un indirizzo IP elastico per Amazon EC2.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la AllocateAddresssezione AWS SDK for JavaScript API Reference.

Associazione di un indirizzo IP elastico a un'istanza

Il seguente esempio di codice mostra come associare un indirizzo IP elastico a un'istanza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la AssociateAddresssezione AWS SDK for JavaScript API Reference.

Creazione di un modello di avvio

L'esempio di codice seguente mostra come creare un modello di avvio di Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    const 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,
        },
      }),
```

- Per i dettagli sull'API, consulta la CreateLaunchTemplatesezione AWS SDK for JavaScript API Reference.

Creazione di un gruppo di sicurezza

Il seguente esempio di codice mostra come creare un gruppo di sicurezza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la CreateSecurityGroupsezione AWS SDK for JavaScript API Reference.

Creazione di una coppia di chiavi di sicurezza

Il seguente esempio di codice mostra come creare una coppia di chiavi di sicurezza per Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la CreateKeyPairsezione AWS SDK for JavaScript API Reference.

Creazione ed esecuzione di un'istanza

Il seguente esempio di codice mostra come creare ed eseguire un'istanza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la RunInstancessezione AWS SDK for JavaScript API Reference.

Eliminare un modello di avvio

L'esempio di codice seguente mostra come eliminare un modello di avvio di Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- Per i dettagli sull'API, consulta la DeleteLaunchTemplatesezione AWS SDK for JavaScript API Reference.

Eliminare un gruppo di sicurezza

Il seguente esempio di codice mostra come eliminare un gruppo di sicurezza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la DeleteSecurityGroupsezione AWS SDK for JavaScript API Reference.

Eliminazione di una coppia di chiavi di sicurezza

Il seguente esempio di codice mostra come eliminare una coppia di chiavi di sicurezza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la DeleteKeyPairsezione AWS SDK for JavaScript API Reference.

Descrivere le regioni

Il seguente esempio di codice mostra come descrivere le regioni Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la DescribeRegionssezione AWS SDK for JavaScript API Reference.

Descrivere le istanze

Il seguente esempio di codice mostra come descrivere le istanze Amazon EC2.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la DescribeInstancessezione AWS SDK for JavaScript API Reference.

Disabilitazione del monitoraggio dettagliato

Il seguente esempio di codice mostra come disabilitare il monitoraggio dettagliato su un'istanza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la UnmonitorInstancessezione AWS SDK for JavaScript API Reference.

Dissociare un indirizzo IP elastico alla sua istanza

Il seguente esempio di codice mostra come dissociare un indirizzo IP elastico da un'istanza Amazon EC2.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la DisassociateAddresssezione AWS SDK for JavaScript API Reference.

Abilitare il monitoraggio

Il seguente esempio di codice mostra come abilitare il monitoraggio per un'istanza Amazon EC2 in esecuzione.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la MonitorInstancessezione AWS SDK for JavaScript API Reference.

Ottenere dati su Amazon Machine Images

Il seguente esempio di codice mostra come ottenere dati su Amazon Machine Images (AMI).

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la DescribeImagessezione AWS SDK for JavaScript API Reference.

Ottenere dati su un gruppo di sicurezza

Il seguente esempio di codice mostra come ottenere dati su un gruppo di sicurezza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la DescribeSecurityGroupssezione AWS SDK for JavaScript API Reference.

Ottenere dati sui tipi di istanze

Il seguente esempio di codice mostra come ottenere dati sui tipi di istanze Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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);
  }
};
```

- Per i dettagli sull'API, consulta la DescribeInstanceTypessezione AWS SDK for JavaScript API Reference.

Ottenimento di dati sul profilo dell'istanza associato a un'istanza

L'esempio di codice seguente mostra come recuperare dati sul profilo dell'istanza associato a un'istanza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    const ec2Client = new EC2Client({});
    const { IamInstanceProfileAssociations } = await ec2Client.send(
      new DescribeIamInstanceProfileAssociationsCommand({
        Filters: [
          { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
        ],
      }),
    );
```

- Per i dettagli sull'API, consulta la DescribeIamInstanceProfileAssociationssezione AWS SDK for JavaScript API Reference.

Ottenere dettagli sugli indirizzi IP elastici

Il seguente esempio di codice mostra come ottenere dettagli sugli indirizzi IP elastici.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la DescribeAddressessezione AWS SDK for JavaScript API Reference.

Ottenimento del VPC predefinito

L'esempio di codice seguente mostra come ottenere il VPC predefinito dell'account corrente.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```

- Per i dettagli sull'API, consulta la DescribeVpcssezione AWS SDK for JavaScript API Reference.

Ottenimento delle sottoreti predefinite per un VPC

L'esempio di codice seguente mostra come ottenere le sottoreti predefinite per un VPC.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const 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"] },
    ],
  }),
```

```
  );
```

- Per i dettagli sull'API, consulta la DescribeSubnetssezione AWS SDK for JavaScript API Reference.

Elencare le coppie di chiavi di sicurezza

Il seguente esempio di codice mostra come elencare le coppie di chiavi di sicurezza di Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la DescribeKeyPairssezione AWS SDK for JavaScript API Reference.

Riavviare un'istanza

Il seguente esempio di codice mostra come riavviare un'istanza Amazon EC2.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la RebootInstancessezione AWS SDK for JavaScript API Reference.

Rilascio di un indirizzo IP elastico

Il seguente esempio di codice mostra come rilasciare un indirizzo IP elastico.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la ReleaseAddresssezione AWS SDK for JavaScript API Reference.

Sostituzione del profilo dell'istanza associato a un'istanza

L'esempio di codice seguente mostra come sostituire il profilo dell'istanza associato a un'istanza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );
```

- Per i dettagli sull'API, consulta la ReplaceIamInstanceProfileAssociationsezione AWS SDK for JavaScript API Reference.

Impostare le regole in entrata per un gruppo di sicurezza

Il seguente esempio di codice mostra come impostare le regole in entrata per un gruppo di sicurezza Amazon EC2.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la AuthorizeSecurityGroupIngresssezione AWS SDK for JavaScript API Reference.

Avviare un'istanza

Il seguente esempio di codice mostra come avviare un'istanza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la StartInstancessezione AWS SDK for JavaScript API Reference.

Arrestare un'istanza

Il seguente esempio di codice mostra come interrompere un'istanza Amazon EC2.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la StopInstancessezione AWS SDK for JavaScript API Reference.

Terminare un'istanza

Il seguente esempio di codice mostra come terminare un'istanza Amazon EC2.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la TerminateInstancessezione AWS SDK for JavaScript API Reference.

Scenari

Creazione e gestione di un servizio resiliente

Il seguente esempio di codice mostra come creare un servizio Web con bilanciamento del carico che restituisca consigli su libri, film e canzoni. L'esempio mostra come il servizio risponde ai guasti e spiega come ristrutturarlo per una maggiore resilienza in caso di guasti.

- Utilizza un gruppo con dimensionamento automatico Amazon EC2 per creare istanze Amazon Elastic Compute Cloud (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze entro un intervallo specificato.
- Gestisci e distribuisci le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo con dimensionamento automatico e inoltra le richieste soltanto alle istanze integre.
- Esegui un server Web Python su ogni istanza EC2 per gestire le richieste HTTP. Il server Web risponde con consigli e controlli dell'integrità.
- Simula un servizio di raccomandazione con una tabella Amazon DynamoDB.
- Controlla la risposta del server web alle richieste e ai controlli sanitari aggiornando AWS Systems Manager i parametri.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui lo scenario interattivo al prompt dei comandi.

```
#!/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);
}
```

Crea passaggi per distribuire tutte le risorse.

```
// 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,
      );
    }
  }),
];
```

Crea i passaggi per eseguire la demo.

```
// 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: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      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;
}
```

Crea i passaggi per distruggere tutte le risorse.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs

- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Nozioni di base sulle istanze

L'esempio di codice seguente mostra come:

- Creare una coppia di chiavi e un gruppo di sicurezza.
- Selezionare un'Amazon Machine Image (AMI) e un tipo di istanza compatibile e quindi creare un'istanza.
- Arrestare e riavviare l'istanza.
- Associazione di un indirizzo IP elastico all'istanza
- Connettiti alla tua istanza con SSH, quindi elimina le risorse.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
import { mkdtempSync, writeFileSync, rmSync } from "fs";
import { tmpdir } from "os";
import { join } from "path";
import { get } from "http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
```

```
  DeleteSecurityGroupCommand,
  DescribeInstancesCommand,
  DescribeKeyPairsCommand,
  DescribeSecurityGroupsCommand,
  DisassociateAddressCommand,
  EC2Client,
  paginateDescribeImages,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";
import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

const ec2Client = new EC2Client();
const ssmClient = new SSMClient();

const prompter = new Prompter();
const confirmMessage = "Continue?";
const tmpDirectory = mkdtempSync(join(tmpdir(), "ec2-scenario-tmp"));

const createKeyPair = async (keyPairName) => {
  // Create a key pair in Amazon EC2.
  const { KeyMaterial, KeyPairId } = await ec2Client.send(
    // A unique name for the key pair. Up to 255 ASCII characters.
    new CreateKeyPairCommand({ KeyName: keyPairName }),
  );

  // Save the private key in a temporary location.
  writeFileSync(`${tmpDirectory}/${keyPairName}.pem`, KeyMaterial, {
    mode: 0o400,
  });

  return KeyPairId;
};
```

```
const describeKeyPair = async (keyPairName) => {
  const command = new DescribeKeyPairsCommand({
    KeyNames: [keyPairName],
  });
  const { KeyPairs } = await ec2Client.send(command);
  return KeyPairs[0];
};

const createSecurityGroup = async (securityGroupName) => {
  const command = new CreateSecurityGroupCommand({
    GroupName: securityGroupName,
    Description: "A security group for the Amazon EC2 example.",
  });
  const { GroupId } = await ec2Client.send(command);
  return GroupId;
};

const allocateIpAddress = async () => {
  const command = new AllocateAddressCommand({});
  const { PublicIp, AllocationId } = await ec2Client.send(command);
  return { PublicIp, AllocationId };
};

const getLocalIpAddress = () => {
  return new Promise((res, rej) => {
    get("http://checkip.amazonaws.com", (response) => {
      let data = "";
      response.on("data", (chunk) => (data += chunk));
      response.on("end", () => res(data.trim()));
    }).on("error", (err) => {
      rej(err);
    });
  });
};

const authorizeSecurityGroupIngress = async (securityGroupId) => {
  const ipAddress = await getLocalIpAddress();
  const command = new AuthorizeSecurityGroupIngressCommand({
    GroupId: securityGroupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
```

```
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  await ec2Client.send(command);
  return ipAddress;
};

const describeSecurityGroup = async (securityGroupName) => {
  const command = new DescribeSecurityGroupsCommand({
    GroupNames: [securityGroupName],
  });
  const { SecurityGroups } = await ec2Client.send(command);

  return SecurityGroups[0];
};

const getAmznLinux2AMIs = async () => {
  const AMIs = [];
  for await (const page of paginateGetParametersByPath(
    {
      client: ssmClient,
    },
    { Path: "/aws/service/ami-amazon-linux-latest" },
  )) {
    page.Parameters.forEach((param) => {
      if (param.Name.includes("amzn2")) {
        AMIs.push(param.Value);
      }
    });
  }

  const imageDetails = [];

  for await (const page of paginateDescribeImages(
    { client: ec2Client },
    { ImageIds: AMIs },
  )) {
    imageDetails.push(...(page.Images || []));
  }

  const choices = imageDetails.map((image, index) => ({
    name: `${image.ImageId} - ${image.Description}`,
```

```
    value: index,
  }));

  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an image.",
    choices,
  });

  return imageDetails[selectedIndex];
};

/**
 * @param {import('@aws-sdk/client-ec2').Image} imageDetails
 */
const getCompatibleInstanceTypes = async (imageDetails) => {
  const paginator = paginateDescribeInstanceTypes(
    { client: ec2Client, pageSize: 25 },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          Values: [imageDetails.Architecture],
        },
        { Name: "instance-type", Values: ["*.micro", "*.small"] },
      ],
    },
  );

  const instanceTypes = [];

  for await (const page of paginator) {
    if (page.InstanceTypes.length) {
      instanceTypes.push(...(page.InstanceTypes || []));
    }
  }

  const choices = instanceTypes.map((type, index) => ({
    name: `${type.InstanceType} - Memory:${type.MemoryInfo.SizeInMiB}`,
    value: index,
  }));
```

```
  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an instance type.",
    choices,
  });
  return instanceTypes[selectedIndex];
};

const runInstance = async ({
  keyPairName,
  securityGroupId,
  imageId,
  instanceType,
}) => {
  const command = new RunInstancesCommand({
    KeyName: keyPairName,
    SecurityGroupIds: [securityGroupId],
    ImageId: imageId,
    InstanceType: instanceType,
    MinCount: 1,
    MaxCount: 1,
  });

  const { Instances } = await ec2Client.send(command);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [Instances[0].InstanceId] },
  );
  return Instances[0].InstanceId;
};

const describeInstance = async (instanceId) => {
  const command = new DescribeInstancesCommand({
    InstanceIds: [instanceId],
  });

  const { Reservations } = await ec2Client.send(command);
  return Reservations[0].Instances[0];
};

const displaySSHConnectionInfo = ({ publicIp, keyPairName }) => {
  return `ssh -i ${tmpDirectory}/${keyPairName}.pem ec2-user@${publicIp}`;
```

```
};

const stopInstance = async (instanceId) => {
  const command = new StopInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(command);
  await waitUntilInstanceStopped(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
};

const startInstance = async (instanceId) => {
  const startCommand = new StartInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(startCommand);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
  return await describeInstance(instanceId);
};

const associateAddress = async ({ allocationId, instanceId }) => {
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  const { AssociationId } = await ec2Client.send(command);
  return AssociationId;
};

const disassociateAddress = async (associationId) => {
  const command = new DisassociateAddressCommand({
    AssociationId: associationId,
  });
  try {
    await ec2Client.send(command);
  } catch (err) {
    console.warn(
      `Failed to disassociated address with association id: ${associationId}`,
      err,
    );
  }
};
```

```
const releaseAddress = async (allocationId) => {
  const command = new ReleaseAddressCommand({
    AllocationId: allocationId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Address with allocation ID ${allocationId} released.\n`);
  } catch (err) {
    console.log(
      `Failed to release address with allocation id: ${allocationId}.`,
      err,
    );
  }
};

const restartInstance = async (instanceId) => {
  console.log("Stopping instance.");
  await stopInstance(instanceId);
  console.log("Instance stopped.");
  console.log("Starting instance.");
  const { PublicIpAddress } = await startInstance(instanceId);
  return PublicIpAddress;
};

const terminateInstance = async (instanceId) => {
  const command = new TerminateInstancesCommand({
    InstanceIds: [instanceId],
  });

  try {
    await ec2Client.send(command);
    await waitUntilInstanceTerminated(
      { client: ec2Client },
      { InstanceIds: [instanceId] },
    );
    console.log(`Instance with ID ${instanceId} terminated.\n`);
  } catch (err) {
    console.warn(`Failed to terminate instance ${instanceId}.`, err);
  }
};

const deleteSecurityGroup = async (securityGroupId) => {
```

```
  const command = new DeleteSecurityGroupCommand({
    GroupId: securityGroupId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Security group ${securityGroupId} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete security group ${securityGroupId}.`, err);
  }
};

const deleteKeyPair = async (keyPairName) => {
  const command = new DeleteKeyPairCommand({
    KeyName: keyPairName,
  });

  try {
    await ec2Client.send(command);
    console.log(`Key pair ${keyPairName} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete key pair ${keyPairName}.`, err);
  }
};

const deleteTemporaryDirectory = () => {
  try {
    rmSync(tmpDirectory, { recursive: true });
    console.log(`Temporary directory ${tmpDirectory} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete temporary directory ${tmpDirectory}.`, err);
  }
};

export const main = async () => {
  const keyPairName = "ec2-scenario-key-pair";
  const securityGroupName = "ec2-scenario-security-group";

  let securityGroupId, ipAllocationId, publicIp, instanceId, associationId;

  console.log(wrapText("Welcome to the Amazon EC2 basic usage scenario."));

  try {
    // Prerequisites
```

```
  console.log(
    "Before you launch an instance, you'll need a few things:",
    "\n - A Key Pair",
    "\n - A Security Group",
    "\n - An IP Address",
    "\n - An AMI",
    "\n - A compatible instance type",
    "\n\n I'll go ahead and take care of the first three, but I'll need your help
for the rest.",
  );

  await prompter.confirm({ message: confirmMessage });

  await createKeyPair(keyPairName);
  securityGroupId = await createSecurityGroup(securityGroupName);
  const { PublicIp, AllocationId } = await allocateIpAddress();
  ipAllocationId = AllocationId;
  publicIp = PublicIp;
  const ipAddress = await authorizeSecurityGroupIngress(securityGroupId);

  const { KeyName } = await describeKeyPair(keyPairName);
  const { GroupName } = await describeSecurityGroup(securityGroupName);
  console.log(`# created the key pair ${KeyName}.\n`);
  console.log(
    `# created the security group ${GroupName}`,
    `and allowed SSH access from ${ipAddress} (your IP).\n`,
  );
  console.log(`# allocated ${publicIp} to be used for your EC2 instance.\n`);

  await prompter.confirm({ message: confirmMessage });

  // Creating the instance
  console.log(wrapText("Create the instance."));
  console.log(
    "You get to choose which image you want. Select an amazon-linux-2 image from
the following:",
  );
  const imageDetails = await getAmznLinux2AMIs();
  const instanceTypeDetails = await getCompatibleInstanceTypes(imageDetails);
  console.log("Creating your instance. This can take a few seconds.");
  instanceId = await runInstance({
    keyPairName,
    securityGroupId,
    imageId: imageDetails.ImageId,
```

```
      instanceType: instanceTypeDetails.InstanceType,
    });
    const instanceDetails = await describeInstance(instanceId);
    console.log(`# instance ${instanceId}.\n`);
    console.log(instanceDetails);
    console.log(
      `\nYou should now be able to SSH into your instance from another terminal:`,
      `\n${displaySSHConnectionInfo({
        publicIp: instanceDetails.PublicIpAddress,
        keyPairName,
      })}`,
    );

    await prompter.confirm({ message: confirmMessage });

    // Understanding the IP address.
    console.log(wrapText("Understanding the IP address."));
    console.log(
      "When you stop and start an instance, the IP address will change. I'll restart
your",
      "instance for you. Notice how the IP address changes.",
    );
    const ipAddressAfterRestart = await restartInstance(instanceId);
    console.log(
      `\n Instance started. The IP address changed from
${instanceDetails.PublicIpAddress} to ${ipAddressAfterRestart}`,
      `\n${displaySSHConnectionInfo({
        publicIp: ipAddressAfterRestart,
        keyPairName,
      })}`,
    );
    await prompter.confirm({ message: confirmMessage });
    console.log(
      `If you want to the IP address to be static, you can associate an allocated`,
      `IP address to your instance. I allocated ${publicIp} for you earlier, and now
I'll associate it to your instance.`,
    );
    associationId = await associateAddress({
      allocationId: ipAllocationId,
      instanceId,
    });
    console.log(
      "Done. Now you should be able to SSH using the new IP.\n",
      `${displaySSHConnectionInfo({ publicIp, keyPairName })}`,
```

```
    );
    await prompter.confirm({ message: confirmMessage });
    console.log(
      "I'll restart the server again so you can see the IP address remains the
 same.",
    );
    const ipAddressAfterAssociated = await restartInstance(instanceId);
    console.log(
      `Done. Here's your SSH info. Notice the IP address hasn't changed.`,
      `\n${displaySSHConnectionInfo({
        publicIp: ipAddressAfterAssociated,
        keyPairName,
      })}`,
    );
    await prompter.confirm({ message: confirmMessage });
  } catch (err) {
    console.error(err);
  } finally {
    // Clean up.
    console.log(wrapText("Clean up."));
    console.log("Now I'll clean up all of the stuff I created.");
    await prompter.confirm({ message: confirmMessage });
    console.log("Cleaning up. Some of these steps can take a bit of time.");
    await disassociateAddress(associationId);
    await terminateInstance(instanceId);
    await releaseAddress(ipAllocationId);
    await deleteSecurityGroup(securityGroupId);
    deleteTemporaryDirectory();
    await deleteKeyPair(keyPairName);
    console.log(
      "Done cleaning up. Thanks for staying until the end!",
      "If you have any feedback please use the feedback button in the docs",
      "or create an issue on GitHub.",
    );
  }
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - AllocateAddress
  - AssociateAddress

- AuthorizeSecurityGroupIngress
- CreateKeyPair
- CreateSecurityGroup
- DeleteKeyPair
- DeleteSecurityGroup
- DescribeImages
- DescribeInstanceTypes
- DescribeInstances
- DescribeKeyPairs
- DescribeSecurityGroups
- DisassociateAddress
- ReleaseAddress
- RunInstances
- StartInstances
- StopInstances
- TerminateInstances
- UnmonitorInstances

Esempi di Elastic Load Balancing con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Elastic Load Balancing.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve Elastic Load Balancing

I seguenti esempi di codice mostrano come iniziare a utilizzare Elastic Load Balancing.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
}
```

- Per i dettagli sull'API, consulta la DescribeLoadBalancerssezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Crea un listener per un sistema di bilanciamento del carico

Il seguente esempio di codice mostra come creare un listener che inoltri le richieste da un sistema di bilanciamento del carico ELB a un gruppo target.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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 },
    ],
  }),
);
```

- Per i dettagli sull'API, consulta la CreateListenersezione AWS SDK for JavaScript API Reference.

Creazione di un gruppo target

Il seguente esempio di codice mostra come creare un gruppo target ELB.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const 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,
  }),
);
```

- Per i dettagli sull'API, consulta la CreateTargetGroupsezione AWS SDK for JavaScript API Reference.

Creazione di un Application Load Balancer

Il seguente esempio di codice mostra come creare un Application Load Balancer ELB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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] },
);
```

- Per i dettagli sull'API, consulta la CreateLoadBalancersezione AWS SDK for JavaScript API Reference.

Eliminazione di un sistema di bilanciamento del carico

Il seguente esempio di codice mostra come eliminare un sistema di bilanciamento del carico ELB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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.");
    }
  });
```

- Per i dettagli sull'API, consulta la DeleteLoadBalancersezione AWS SDK for JavaScript API Reference.

Eliminazione di un gruppo target

Il seguente esempio di codice mostra come eliminare un gruppo target ELB.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const 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;
  }
```

- Per i dettagli sull'API, consulta la DeleteTargetGroupsezione AWS SDK for JavaScript API Reference.

Descrivi i gruppi target

Il seguente esempio di codice mostra come descrivere gruppi target specifici.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- Per i dettagli sull'API, consulta la DescribeTargetGroupssezione AWS SDK for JavaScript API Reference.

Ottieni l'endpoint di un sistema di bilanciamento del carico

Il seguente esempio di codice mostra come ottenere l'endpoint di un sistema di bilanciamento del carico ELB.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
}
```

- Per i dettagli sull'API, consulta la DescribeLoadBalancerssezione AWS SDK for JavaScript API Reference.

Ottieni lo stato di salute di un gruppo target

Il seguente esempio di codice mostra come ottenere lo stato delle istanze in un gruppo target ELB.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- Per i dettagli sull'API, consulta la DescribeTargetHealthsezione AWS SDK for JavaScript API Reference.

Scenari

Creazione e gestione di un servizio resiliente

Il seguente esempio di codice mostra come creare un servizio Web con bilanciamento del carico che restituisca consigli su libri, film e canzoni. L'esempio mostra come il servizio risponde ai guasti e spiega come ristrutturarlo per una maggiore resilienza in caso di guasti.

- Utilizza un gruppo con dimensionamento automatico Amazon EC2 per creare istanze Amazon Elastic Compute Cloud (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze entro un intervallo specificato.
- Gestisci e distribuisci le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo con dimensionamento automatico e inoltra le richieste soltanto alle istanze integre.
- Esegui un server Web Python su ogni istanza EC2 per gestire le richieste HTTP. Il server Web risponde con consigli e controlli dell'integrità.

- Simula un servizio di raccomandazione con una tabella Amazon DynamoDB.
- Controlla la risposta del server web alle richieste e ai controlli sanitari aggiornando AWS Systems Manager i parametri.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui lo scenario interattivo al prompt dei comandi.

```
#!/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);
}
```

Crea passaggi per distribuire tutte le risorse.

```
// 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,
      );
    }
  }),
];
```

Crea i passaggi per eseguire la demo.

```
// 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: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
```

```
      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;
}
```

Crea i passaggi per distruggere tutte le risorse.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers

- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

EventBridge esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. EventBridge

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Aggiunta di una destinazione

Il seguente esempio di codice mostra come aggiungere un target a un EventBridge evento Amazon.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per i dettagli sull'API, consulta la PutTargetssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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);
```

```
  }
});
```

- Per i dettagli sull'API, consulta la PutTargetssezione AWS SDK for JavaScript API Reference.

Creazione di una regola

Il seguente esempio di codice mostra come creare una EventBridge regola Amazon.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per i dettagli sull'API, consulta la PutRulesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per i dettagli sull'API, consulta la PutRulesezione AWS SDK for JavaScript API Reference.

Invio di eventi

Il seguente esempio di codice mostra come inviare EventBridge eventi Amazon.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per i dettagli sull'API, consulta la PutEventssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per i dettagli sull'API, consulta la PutEventssezione AWS SDK for JavaScript API Reference.

AWS Glue esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. AWS Glue

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve AWS Glue

L'esempio di codice seguente mostra come iniziare a utilizzare AWS Glue.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
};
```

- Per i dettagli sull'API, ListJobsconsulta AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Creazione di un crawler

Il seguente esempio di codice mostra come creare un AWS Glue crawler.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
};
```

- Per i dettagli sull'API, CreateCrawlerconsulta AWS SDK for JavaScript API Reference.

Creazione di una definizione di processo

Il seguente esempio di codice mostra come creare una definizione di AWS Glue processo.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const 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);
};
```

- Per i dettagli sull'API, CreateJobconsulta AWS SDK for JavaScript API Reference.

Eliminazione di un crawler

Il seguente esempio di codice mostra come eliminare un AWS Glue crawler.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, DeleteCrawlerconsulta AWS SDK for JavaScript API Reference.

Eliminazione di un database dal catalogo dati

Il seguente esempio di codice mostra come eliminare un database da AWS Glue Data Catalog.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, DeleteDatabaseconsulta AWS SDK for JavaScript API Reference.

Eliminazione di una definizione di processo

Il seguente esempio di codice mostra come eliminare una definizione di AWS Glue processo e tutte le esecuzioni associate.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, DeleteJobconsulta AWS SDK for JavaScript API Reference.

Eliminazione di una tabella da un database

Il seguente esempio di codice mostra come eliminare una tabella da un AWS Glue Data Catalog database.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK for JavaScript API Reference.

Ottenimento di un crawler

Il seguente esempio di codice mostra come ottenere un AWS Glue crawler.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, GetCrawlerconsulta AWS SDK for JavaScript API Reference.

Ottenimento di un database dal catalogo dati

Il seguente esempio di codice mostra come ottenere un database da AWS Glue Data Catalog.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getDatabase = (name) => {
  const client = new GlueClient({});
```

```
  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, GetDatabaseconsulta AWS SDK for JavaScript API Reference.

Ottenimento dell'esecuzione di un processo

Il seguente esempio di codice mostra come eseguire un AWS Glue job.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, GetJobRunconsulta AWS SDK for JavaScript API Reference.

Ottenimento di database dal catalogo dati

L'esempio di codice seguente mostra come ottenere un elenco di database da AWS Glue Data Catalog.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- Per i dettagli sull'API, GetDatabasesconsulta AWS SDK for JavaScript API Reference.

Ottenimento di un processo dal catalogo dati

L'esempio di codice seguente mostra come ottenere un processo da AWS Glue Data Catalog.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, GetJobconsulta AWS SDK for JavaScript API Reference.

Ottenimento di esecuzioni di un processo

Il seguente esempio di codice mostra come eseguire un AWS Glue job.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, GetJobRunsconsulta AWS SDK for JavaScript API Reference.

Ottenimento di tabelle da un database

Il seguente esempio di codice mostra come ottenere tabelle da un database in AWS Glue Data Catalog.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, GetTablesconsulta AWS SDK for JavaScript API Reference.

Elencazione delle definizioni di processo

Il seguente esempio di codice mostra come elencare le definizioni dei AWS Glue processi.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- Per i dettagli sull'API, ListJobsconsulta AWS SDK for JavaScript API Reference.

Avvio di un crawler

Il seguente esempio di codice mostra come avviare un AWS Glue crawler.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, StartCrawlerconsulta AWS SDK for JavaScript API Reference.

Avviare un'esecuzione del processo

Il seguente esempio di codice mostra come avviare l'esecuzione di un AWS Glue job.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const 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);
};
```

- Per i dettagli sull'API, StartJobRunconsulta AWS SDK for JavaScript API Reference.

Scenari

Nozioni di base su crawler e processi

L'esempio di codice seguente mostra come:

- Crea un crawler che esegue la scansione di un bucket Amazon S3 pubblico e genera un database di metadati in formato CSV.
- Elenca le informazioni su database e tabelle nel tuo AWS Glue Data Catalog.
- Crea un processo per estrarre i dati CSV dal bucket S3, trasformare i dati e caricare l'output in formato JSON in un altro bucket S3.
- Elenca le informazioni sulle esecuzioni dei processi, visualizza i dati trasformati e pulisci le risorse.

Per ulteriori informazioni, consulta Tutorial: Guida introduttiva a AWS Glue Studio.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare e avviare un crawler in grado di eseguire il crawling di un bucket pubblico di Amazon Simple Storage Service (Amazon S3) generando un database di metadati che descrive i dati rilevati in formato CSV.

```
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;
  }
};
```

```
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 };
  };
```

Elenca le informazioni su database e tabelle nel tuo 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 };
  };
```

```
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
```

Creare e avviare un processo che estrae i dati CSV dal bucket Amazon S3 di origine, li trasforma rimuovendo e rinominando i campi e carica l'output in formato JSON in un altro bucket 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 };
  };
```

Elencare le informazioni sulle esecuzioni dei processi e visualizzare alcuni dei dati trasformati.

```
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);
};

const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (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 };
  };
```

Eliminare tutte le risorse create dalla demo.

```
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);
};

const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  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" });
  }
};

const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error)
    )
```

```
  );

const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null })
    );

    if (TableList && TableList.length > 0) {
      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 };
  };

const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error))
  );

const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      const { dbNames } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
```

```
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbNames.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
      }
    }

    return { ...context };
  };

const cleanUpCrawlerStep = async (context) => {
  log(`Deleting crawler.`);

  try {
    await deleteCrawler(process.env.CRAWLER_NAME);
    log("Crawler deleted.", { type: "success" });
  } catch (err) {
    if (err.name === "EntityNotFoundException") {
      log(`Crawler is already deleted.`);
    } else {
      throw err;
    }
  }

  return { ...context };
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable

- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

HealthImaging esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. HealthImaging

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve HealthImaging

L'esempio di codice seguente mostra come iniziare a utilizzare HealthImaging.

SDK per 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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListDatastoresReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Argomenti

- Azioni
- Scenari

Azioni

Aggiungere un tag a una risorsa

Il seguente esempio di codice mostra come aggiungere un tag a una HealthImaging risorsa.

SDK per 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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API TagResourceReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Copia un set di immagini

Il seguente esempio di codice mostra come copiare un set di HealthImaging immagini.

SDK per JavaScript (v3)

Funzione di utilità per copiare un set di immagini.

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
```

```
  //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //      },
  //       datastoreId: 'xxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING',
  //             latestVersionId: '1',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //       },
  //       sourceImageSetProperties: {
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

Copia un set di immagini senza una destinazione.

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

Copia un set di immagini con una destinazione.

```
try {
  await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "4",
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.error(err);
}
```

- Per i dettagli sull'API, consulta la CopyImageSetsezione AWS SDK for JavaScript API Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un archivio dati

Il seguente esempio di codice mostra come creare un HealthImaging data store.

SDK per JavaScript (v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName })
  );
```

```
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //    datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API CreateDatastoreReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare un archivio dati

Il seguente esempio di codice mostra come eliminare un HealthImaging data store.

SDK per JavaScript (v3)

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId })
  );
```

```
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //         },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteDatastoreReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare un set di immagini

Il seguente esempio di codice mostra come eliminare un set di HealthImaging immagini.

SDK per 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 = "xxxxxxxxxxxxxxxx",
```

```
  imageSetId = "xxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteImageSetReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Procurati una cornice per l'immagine

Il seguente esempio di codice mostra come ottenere una cornice per l'immagine.

SDK per 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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [GetImageFrame](#)Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Ottieni le proprietà del data store

Il seguente esempio di codice mostra come ottenere le proprietà dell'archivio HealthImaging dati.

SDK per JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
```

```
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response["datastoreProperties"];
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetDatastoreReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottieni le proprietà del set di immagini

Il seguente esempio di codice mostra come ottenere le proprietà del set di HealthImaging immagini.

SDK per JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
```

```
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetImageSetReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Importa le proprietà del lavoro

Il seguente esempio di codice mostra come ottenere le proprietà del lavoro di importazione.

SDK per JavaScript (v3)

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
```

```
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta getDICOM ImportJob in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottieni i metadati per un set di immagini

Il seguente esempio di codice mostra come ottenere i metadati per un set di HealthImaging immagini.

SDK per JavaScript (v3)

Funzione di utilità per ottenere i metadati del set di immagini.

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
```

```
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

Ottieni i metadati del set di immagini senza versione.

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

Ottieni i metadati del set di immagini con la versione.

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.log("Error", err);
```

```
  }
```

- Per i dettagli sull'API, consulta la sezione GetImageSetMetadata AWS SDK for JavaScriptAPI Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Importa dati di massa in un archivio dati

Il seguente esempio di codice mostra come importare dati in blocco in un HealthImaging data store.

SDK per 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;
};
```

- Per i dettagli sull'API, consulta StartDicom ImportJob in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca gli archivi dati

Il seguente esempio di codice mostra come elencare gli archivi HealthImaging dati.

SDK per JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
```

```
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListDatastoresReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le versioni dei set di immagini

Il seguente esempio di codice mostra come elencare le versioni dei set di HealthImaging immagini.

SDK per JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
```

```
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListImageSetVersionsReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i lavori di importazione per un archivio dati

Il seguente esempio di codice mostra come elencare i processi di importazione per un HealthImaging data store.

SDK per JavaScript (v3)

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
```

```
/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
```

```
};
```

- Per i dettagli sull'API, consulta ListDicom ImportJobs in API Reference.AWS SDK for JavaScript

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elencazione dei tag associati a una risorsa

Il seguente esempio di codice mostra come elencare i tag di una HealthImaging risorsa.

SDK per 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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListTagsForResourceReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Rimuovere un tag da una risorsa

Il seguente esempio di codice mostra come rimuovere un tag da una HealthImaging risorsa.

SDK per 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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API UntagResourceReference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Cerca set di immagini

Il seguente esempio di codice mostra come cercare set di HealthImaging immagini.

SDK per JavaScript (v3)

La funzione di utilità per la ricerca di set di immagini.

```
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.
 */
export const searchImageSets = async (
  datastoreId = "xxxxxxxx",
  filters = []
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };
```

```
  const commandParams = {
    datastoreId: datastoreId,
    searchCriteria: {
      filters,
    },
  };

  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 d'uso #1: operatore EQUAL.

```
  const datastoreId = "12345678901234567890123456789012";
```

```
try {
  const filters = [
    {
      values: [{ DICOMPatientId: "9227465" }],
      operator: "EQUAL",
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}
```

Caso d'uso #2: operatore BETWEEN che utilizza DICOM StudyDate e DICOMStudyTime.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "19900101",
            DICOMStudyTime: "000000",
          },
        },
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "20230901",
            DICOMStudyTime: "000000",
          },
        },
      ],
      operator: "BETWEEN",
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
```

```
}
```

Caso d'uso #3: operatore BETWEEN che utilizza CreateDat. Gli studi sul tempo erano stati precedentemente proseguiti.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [
        { createdAt: new Date("1985-04-12T23:20:50.52Z") },
        { createdAt: new Date("2023-09-12T23:20:50.52Z") },
      ],
      operator: "BETWEEN",
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}
```

- Per i dettagli sulle API, consulta la sezione SearchImageSets AWS SDK for JavaScriptAPI Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna i metadati del set di immagini

Il seguente esempio di codice mostra come aggiornare i metadati del set di HealthImaging immagini.

SDK per JavaScript (v3)

```
import {UpdateImageSetMetadataCommand} from "@aws-sdk/client-medical-imaging";
```

```
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}') => {
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
            imageSetId: imageSetId,
            latestVersionId: latestVersionId,
            updateImageSetMetadataUpdates: updateMetadata
        })
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
};
```

Codifica i metadati.

```
    const updatableAttributes =
JSON.stringify({
  "SchemaVersion": 1.1,
  "Patient": {
    "DICOM": {
      "PatientName": "Garcia^Gloria"
    }
  }
})

    const updateMetadata = {
        "DICOMUpdates": {
            "updatableAttributes":
                new TextEncoder().encode(updatableAttributes)
        }
    };

    await updateImageSetMetadata("12345678901234567890123456789012",
 "12345678901234567890123456789012",
        "1", updateMetadata);
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [UpdateImageSetMetadata](#)Reference.

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Scenari

Taggare un archivio dati

Il seguente esempio di codice mostra come etichettare un HealthImaging data store.

SDK per JavaScript (v3)

Per etichettare un archivio dati.

```
  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);
  }
```

La funzione di utilità per etichettare una risorsa.

```
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;
};
```

Per elencare i tag per un archivio dati.

```
  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 funzione di utilità per elencare i tag di una risorsa.

```
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;
};
```

Per rimuovere i tag da un archivio dati.

```
  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 funzione di utilità per rimuovere il tag di una risorsa.

```
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;
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Taggare un set di immagini

Il seguente esempio di codice mostra come etichettare un set di HealthImaging immagini.

SDK per JavaScript (v3)

Per etichettare un set di immagini.

```
  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);
```

```
  }
```

La funzione di utilità per etichettare una risorsa.

```
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;
};
```

Per elencare i tag per un set di immagini.

```
  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 funzione di utilità per elencare i tag di una risorsa.

```
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;
};
```

Per rimuovere i tag da un set di immagini.

```
  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 funzione di utilità per rimuovere il tag di una risorsa.

```
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;
```

```
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esempi IAM che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con IAM.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello IAM

Gli esempi di codice seguenti mostrano come iniziare a utilizzare IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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.`);
};
```

- Per i dettagli sull'API, ListPoliciesconsulta AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Collegamento di una policy a un ruolo

Il seguente esempio di codice mostra come collegare una policy IAM a un ruolo.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Collega la policy.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, AttachRolePolicyconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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");
```

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, AttachRolePolicyconsulta AWS SDK for JavaScript API Reference.

Collegamento di una policy inline a un ruolo

Gli esempi di codice seguenti mostrano come aggiungere una policy inline a un ruolo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
};
```

- Per i dettagli sull'API, PutRolePolicyconsulta AWS SDK for JavaScript API Reference.

Come creare un provider SAML

Il seguente esempio di codice mostra come creare un provider SAML AWS Identity and Access Management (IAM).

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
```

```
};
```

- Per informazioni dettagliate sull'API, consulta CreateSAMLProvider nella Documentazione di riferimento dell'API di AWS SDK for JavaScript .

Creazione di un gruppo

L'esempio di codice seguente mostra come creare un gruppo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
};
```

- Per i dettagli sull'API, CreateGroupconsulta AWS SDK for JavaScript API Reference.

Creazione di una policy

Il seguente esempio di codice mostra come creare una policy IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea la policy.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreatePolicyconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreatePolicyconsulta AWS SDK for JavaScript API Reference.

Creare un ruolo

Il seguente esempio di codice mostra come creare un ruolo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il ruolo.

```
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);
};
```

- Per i dettagli sull'API, CreateRoleconsulta AWS SDK for JavaScript API Reference.

Creazione di un ruolo collegato ai servizi

Il seguente esempio di codice mostra come creare un ruolo collegato a un servizio IAM.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un ruolo collegato ai servizi.

```
import { CreateServiceLinkedRoleCommand, 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,
  });

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

- Per i dettagli sull'API, CreateServiceLinkedRoleconsulta AWS SDK for JavaScript API Reference.

Creazione di un utente

Il seguente esempio di codice mostra come creare un utente IAM.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare l'utente.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateUserconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateUserconsulta AWS SDK for JavaScript API Reference.

Creare una chiave di accesso

Il seguente esempio di codice mostra come creare una chiave di accesso IAM.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea la chiave di accesso.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateAccessKeyconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateAccessKeyconsulta AWS SDK for JavaScript API Reference.

Creazione di un alias per un account

Il seguente esempio di codice mostra come creare un alias per un account IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea l'alias dell'account.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateAccountAliasconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateAccountAliasconsulta AWS SDK for JavaScript API Reference.

Creazione di un profilo dell'istanza

Il seguente esempio di codice mostra come creare un profilo dell'istanza IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
```

```
  );
```

- Per i dettagli sull'API, CreateInstanceProfileconsulta AWS SDK for JavaScript API Reference.

Come eliminare un provider SAML

Il seguente esempio di codice mostra come eliminare un provider SAML AWS Identity and Access Management (IAM).

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
};
```

- Per informazioni dettagliate sull'API, consulta DeleteSAMLProvider nella Documentazione di riferimento dell'API di AWS SDK for JavaScript .

Eliminazione di un gruppo

L'esempio di codice seguente mostra come eliminare un gruppo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
};
```

- Per i dettagli sull'API, DeleteGroupconsulta AWS SDK for JavaScript API Reference.

Eliminazione di una policy

Il seguente esempio di codice mostra come eliminare una policy IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare il criterio.

```
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);
};
```

- Per i dettagli sull'API, DeletePolicyconsulta AWS SDK for JavaScript API Reference.

Eliminazione di un ruolo

Il seguente esempio di codice mostra come eliminare un ruolo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina il ruolo.

```
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);
};
```

- Per i dettagli sull'API, DeleteRoleconsulta AWS SDK for JavaScript API Reference.

Eliminazione di una policy del ruolo

L'esempio di codice seguente mostra come eliminare una policy del ruolo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
};
```

- Per i dettagli sull'API, DeleteRolePolicyconsulta AWS SDK for JavaScript API Reference.

Eliminazione di un certificato del server

L'esempio di codice seguente mostra come eliminare un certificato del server IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina un certificato del server.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteServerCertificateconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteServerCertificateconsulta AWS SDK for JavaScript API Reference.

Eliminazione di un ruolo collegato ai servizi

Il seguente esempio di codice mostra come eliminare un ruolo collegato a un servizio IAM.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
};
```

- Per i dettagli sull'API, DeleteServiceLinkedRoleconsulta AWS SDK for JavaScript API Reference.

Eliminazione di un utente

Il seguente esempio di codice mostra come eliminare un utente IAM.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare l'utente.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteUserconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteUserconsulta AWS SDK for JavaScript API Reference.

Eliminare una chiave di accesso

Il seguente esempio di codice mostra come eliminare una chiave di accesso IAM.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina la chiave di accesso.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteAccessKeyconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteAccessKeyconsulta AWS SDK for JavaScript API Reference.

Eliminazione di un alias di un account

Il seguente esempio di codice mostra come eliminare l'alias di un account IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina l'alias dell'account.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteAccountAliasconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteAccountAliasconsulta AWS SDK for JavaScript API Reference.

Eliminazione di un profilo dell'istanza IAM

Il seguente esempio di codice mostra come eliminare un profilo dell'istanza IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const client = new IAMClient({});
await client.send(
  new DeleteInstanceProfileCommand({
    InstanceProfileName: NAMES.instanceProfileName,
  }),
);
```

- Per i dettagli sull'API, DeleteInstanceProfileconsulta AWS SDK for JavaScript API Reference.

Scollegamento di una policy da un ruolo

Il seguente esempio di codice mostra come scollegare una policy IAM da un ruolo.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Scollega la policy.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DetachRolePolicyconsulta AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
          }
        });
      }
    });
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DetachRolePolicysezione AWS SDK for JavaScript API Reference.

Ottenere una policy

Il seguente esempio di codice mostra come ottenere una policy IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la policy.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetPolicysezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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);
```

```
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetPolicysezione AWS SDK for JavaScript API Reference.

Recupero di un ruolo

Il seguente esempio di codice mostra come ottenere un ruolo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera il ruolo.

```
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);
};
```

- Per i dettagli sull'API, consulta la GetRolesezione AWS SDK for JavaScript API Reference.

Recupero di un certificato del server

L'esempio di codice seguente mostra come recuperare un certificato del server IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera un certificato del server.

```
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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetServerCertificatesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetServerCertificatesezione AWS SDK for JavaScript API Reference.

Come ottenere uno stato di eliminazione di un ruolo collegato a un servizio

Il seguente esempio di codice mostra come ottenere lo stato di eliminazione di un ruolo collegato al servizio AWS Identity and Access Management (IAM).

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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);
};
```

- Per i dettagli sull'API, consulta la GetServiceLinkedRoleDeletionStatussezione AWS SDK for JavaScript API Reference.

Recupero dei dati sull'ultimo utilizzo di una chiave di accesso

Il seguente esempio di codice mostra come ottenere dati sull'ultimo utilizzo di una chiave di accesso IAM.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la chiave di accesso.

```
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;
```

```
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetAccessKeyLastUsedsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetAccessKeyLastUsedsezione AWS SDK for JavaScript API Reference.

Recupero della policy sulla password dell'account

Il seguente esempio di codice mostra come ottenere la politica relativa alle password degli account IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la policy sulla password dell'account.

```
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;
};
```

- Per i dettagli sull'API, consulta la GetAccountPasswordPolicysezione AWS SDK for JavaScript API Reference.

Elencare gli IdP SAML

Il seguente esempio di codice mostra come elencare i provider SAML per IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca gli IdP 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;
};
```

- Per informazioni dettagliate sull'API, consulta la sezione ListSAMLProviders nella Documentazione di riferimento dell'API AWS SDK for JavaScript .

Elencare le chiavi di accesso di un utente

Il seguente esempio di codice mostra come elencare le chiavi di accesso IAM di un utente.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le chiavi di accesso.

```
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;
    }
  }
}
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListAccessKeyssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListAccessKeyssezione AWS SDK for JavaScript API Reference.

Elencare gli alias di un account

Il seguente esempio di codice mostra come elencare gli alias degli account IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca gli alias di un account.

```
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;
    }
  }
}
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListAccountAliasessezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per i dettagli sull'API, consulta la ListAccountAliasessezione AWS SDK for JavaScript API Reference.

Elencare i gruppi

Il seguente esempio di codice mostra come elencare i gruppi IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i gruppi.

```
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;
    }
  }
}
```

- Per i dettagli sull'API, consulta la ListGroupssezione AWS SDK for JavaScript API Reference.

Elencare le policy inline per un ruolo

Il seguente esempio di codice mostra come elencare le politiche in linea per un ruolo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le policy.

```
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;
    }
  }
}
```

- Per i dettagli sull'API, consulta la ListRolePoliciessezione AWS SDK for JavaScript API Reference.

Elencare le policy

Il seguente esempio di codice mostra come elencare le politiche IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le policy.

```
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;
    }
  }
}
```

- Per i dettagli sull'API, consulta la ListPoliciessezione AWS SDK for JavaScript API Reference.

Elencare le policy collegate a un ruolo

Il seguente esempio di codice mostra come elencare le policy associate a un ruolo IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le policy collegate a un ruolo.

```
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;
    }
  }
}
```

- Per i dettagli sull'API, consulta la ListAttachedRolePoliciessezione AWS SDK for JavaScript API Reference.

Elencare i ruoli

Il seguente esempio di codice mostra come elencare i ruoli IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i ruoli.

```
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;
    }
  }
}
```

- Per i dettagli sull'API, consulta la ListRolessezione AWS SDK for JavaScript API Reference.

Elencare i certificati del server

L'esempio di codice seguente mostra come elencare tutti i certificati del server IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i certificati.

```
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;
    }
  }
}
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListServerCertificatessezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListServerCertificatessezione AWS SDK for JavaScript API Reference.

Elencare gli utenti

Il seguente esempio di codice mostra come elencare gli utenti IAM.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca gli utenti.

```
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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListUserssezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListUserssezione AWS SDK for JavaScript API Reference.

Aggiornamento di un certificato del server

L'esempio di codice seguente mostra come aggiornare un certificato del server IAM.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna un certificato del server.

```
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);
```

```
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la UpdateServerCertificatesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la UpdateServerCertificatesezione AWS SDK for JavaScript API Reference.

Aggiornamento di un utente

Il seguente esempio di codice mostra come aggiornare un utente IAM.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna l'utente.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per i dettagli sull'API, consulta la UpdateUsersezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la UpdateUsersezione AWS SDK for JavaScript API Reference.

Aggiornamento di una chiave di accesso

Il seguente esempio di codice mostra come aggiornare una chiave di accesso IAM.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna la chiave di accesso.

```
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);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la UpdateAccessKeysezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la UpdateAccessKeysezione AWS SDK for JavaScript API Reference.

Come caricare un certificato del server

Il seguente esempio di codice mostra come caricare un certificato del server AWS Identity and Access Management (IAM).

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
};
```

- Per i dettagli sull'API, consulta la UploadServerCertificatesezione AWS SDK for JavaScript API Reference.

Scenari

Creazione e gestione di un servizio resiliente

Il seguente esempio di codice mostra come creare un servizio Web con bilanciamento del carico che restituisca consigli su libri, film e canzoni. L'esempio mostra come il servizio risponde ai guasti e spiega come ristrutturarlo per una maggiore resilienza in caso di guasti.

- Utilizza un gruppo con dimensionamento automatico Amazon EC2 per creare istanze Amazon Elastic Compute Cloud (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze entro un intervallo specificato.
- Gestisci e distribuisci le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo con dimensionamento automatico e inoltra le richieste soltanto alle istanze integre.

- Esegui un server Web Python su ogni istanza EC2 per gestire le richieste HTTP. Il server Web risponde con consigli e controlli dell'integrità.
- Simula un servizio di raccomandazione con una tabella Amazon DynamoDB.
- Controlla la risposta del server web alle richieste e ai controlli sanitari aggiornando AWS Systems Manager i parametri.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui lo scenario interattivo al prompt dei comandi.

```
#!/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);
}
```

Crea passaggi per distribuire tutte le risorse.

```
// 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,
      );
    }
  }),
];
```

Crea i passaggi per eseguire la demo.

```
// 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: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
```

```
  {
    whileConfig: {
      inputEquals: true,
      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;
}
```

Crea i passaggi per distruggere tutte le risorse.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones

- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Creazione di un utente e assunzione di un ruolo

Il seguente esempio di codice mostra come creare un utente e assumere un ruolo.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

- Crea un utente che non disponga di autorizzazioni.
- Crea un ruolo che conceda l'autorizzazione per elencare i bucket Amazon S3 per l'account.
- Aggiungi una policy per consentire all'utente di assumere il ruolo.
- Assumi il ruolo ed elenca i bucket S3 utilizzando le credenziali temporanee, quindi ripulisci le risorse.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un utente IAM che conceda l'autorizzazione per elencare i bucket Amazon S3. L'utente dispone dei diritti soltanto per assumere il ruolo. Dopo aver assunto il ruolo, utilizza le credenziali temporanee per elencare i bucket per l'account.

```
import {
  CreateUserCommand,
  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";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

export const main = async () => {
  // Create a user. The user has no permissions by default.
  const { User } = await iamClient.send(
    new CreateUserCommand({ UserName: userName }),
  );

  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"));
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Esempi di Lambda con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Lambda.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Lambda

L'esempio di codice seguente mostra come iniziare a utilizzare Lambda.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
```

```
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Creazione di una funzione

Il seguente esempio di codice mostra come creare una funzione Lambda.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
```

```
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK for JavaScript API Reference.

Eliminare una funzione

Il seguente esempio di codice mostra come eliminare una funzione Lambda.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK for JavaScript API Reference.

Ottenimento di una funzione

L'esempio di codice seguente mostra come ottenere una funzione Lambda.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK for JavaScript API Reference.

Richiamo di una funzione

Il seguente esempio di codice mostra come richiamare una funzione Lambda.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
```

```
  return { logs, result };
};
```

- Per informazioni dettagliate sulle API, consulta Invoke nella Documentazione di riferimento delle API AWS SDK for JavaScript .

Elencare le funzioni

L'esempio di codice seguente mostra come elencare le funzioni Lambda.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK for JavaScript API Reference.

Aggiornamento del codice della funzione

L'esempio di codice seguente mostra come aggiornare il codice della funzione Lambda.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK for JavaScript API Reference.

Aggiornamento della configurazione della funzione

L'esempio di codice seguente mostra come aggiornare la configurazione della funzione Lambda.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK for JavaScript API Reference.

Scenari

Nozioni di base sulle funzioni

L'esempio di codice seguente mostra come:

- Crea un ruolo IAM e una funzione Lambda, quindi carica il codice del gestore.
- Richiamare la funzione con un singolo parametro e ottenere i risultati.
- Aggiorna il codice della funzione e configuralo con una variabile di ambiente.
- Richiamare la funzione con nuovi parametri e ottenere i risultati. Visualizza il log di esecuzione restituito.
- Elenca le funzioni dell'account, quindi elimina le risorse.

Per ulteriori informazioni sull'utilizzo di Lambda, consulta Creare una funzione Lambda con la console.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un ruolo AWS Identity and Access Management (IAM) che conceda a Lambda l'autorizzazione di scrittura nei log.

```
    log(`Creating role (${NAME_ROLE_LAMBDA})...`);
    const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

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

```
/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

Creare una funzione Lambda e caricare il codice del gestore.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

Richiamare la funzione con un singolo parametro e ottenere i risultati.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
```

```
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

Aggiornare il codice della funzione e configurare il suo ambiente Lambda con una variabile di ambiente.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

Elencare le funzioni per l'account.

```
const listFunctions = () => {
  const client = new LambdaClient({});
```

```
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Eliminare il ruolo IAM e la funzione Lambda.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Esempi di Amazon Personalize utilizzando SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Personalize.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

Crea un processo di interfaccia batch

Il seguente esempio di codice mostra come creare un processo di interfaccia batch Amazon Personalize.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateBatchInferenceJobsezione AWS SDK for JavaScript API Reference.

Crea un processo di segmentazione in batch

Il seguente esempio di codice mostra come creare un processo di segmentazione batch di Amazon Personalize.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateBatchSegmentJobsezione AWS SDK for JavaScript API Reference.

Creazione di una campagna

Il seguente esempio di codice mostra come creare una campagna Amazon Personalize.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateCampaignsezione AWS SDK for JavaScript API Reference.

Crea un set di dati

Il seguente esempio di codice mostra come creare un set di dati Amazon Personalize.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateDatasetsezione AWS SDK for JavaScript API Reference.

Crea un processo di esportazione del set di dati

Il seguente esempio di codice mostra come creare un processo di esportazione di set di dati Amazon Personalize.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateDatasetExportJobsezione AWS SDK for JavaScript API Reference.

Crea un gruppo di set di dati

Il seguente esempio di codice mostra come creare un gruppo di set di dati Amazon Personalize.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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);
```

Crea un gruppo di set di dati di dominio.

```
// 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();
```

- Per i dettagli sull'API, consulta CreateDatasetGroup AWS SDK for JavaScriptAPI Reference.

Crea un processo di importazione del set di dati

Il seguente esempio di codice mostra come creare un processo di importazione di set di dati Amazon Personalize.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateDatasetImportJobsezione AWS SDK for JavaScript API Reference.

Crea uno schema di dominio

Il seguente esempio di codice mostra come creare uno schema di dominio Amazon Personalize.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateSchemasezione AWS SDK for JavaScript API Reference.

Crea un filtro

Il seguente esempio di codice mostra come creare un filtro Amazon Personalize.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateFiltersezione AWS SDK for JavaScript API Reference.

Crea un programma di raccomandazione

Il seguente esempio di codice mostra come creare un programma di raccomandazione Amazon Personalize.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateRecommendersezione AWS SDK for JavaScript API Reference.

Crea uno schema

Il seguente esempio di codice mostra come creare uno schema Amazon Personalize.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateSchemasezione AWS SDK for JavaScript API Reference.

Crea una soluzione

Il seguente esempio di codice mostra come creare una soluzione Amazon Personalize.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateSolutionsezione AWS SDK for JavaScript API Reference.

Crea una versione della soluzione

Il seguente esempio di codice mostra come creare una soluzione Amazon Personalize.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateSolutionVersionsezione AWS SDK for JavaScript API Reference.

Crea un tracker di eventi

Il seguente esempio di codice mostra come creare un tracker di eventi Amazon Personalize.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la CreateEventTrackersezione AWS SDK for JavaScript API Reference.

Esempi di Amazon Personalize Events utilizzando SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Personalize Events.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

Importa elementi in un set di dati

Il seguente esempio di codice mostra come importare in modo incrementale gli articoli in un set di dati Amazon Personalize Events.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la PutItemssezione AWS SDK for JavaScript API Reference.

Importa i dati degli eventi di interazione in tempo reale

Il seguente esempio di codice mostra come importare dati di eventi di interazione in tempo reale in Amazon Personalize Events.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la PutEventssezione AWS SDK for JavaScript API Reference.

Importa un utente in modo incrementale

Il seguente esempio di codice mostra come importare in modo incrementale un utente in Amazon Personalize Events Events.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la PutUserssezione AWS SDK for JavaScript API Reference.

Esempi di Amazon Personalize Runtime utilizzando SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Personalize Runtime.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Ottieni consigli (gruppo di set di dati personalizzato)

Il seguente esempio di codice mostra come ottenere i consigli classificati di Amazon Personalize Runtime Runtime.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

- Per i dettagli sull'API, consulta la GetPersonalizedRankingsezione AWS SDK for JavaScript API Reference.

Ottieni consigli da un consulente (gruppo di set di dati del dominio)

Il seguente esempio di codice mostra come ottenere consigli su Amazon Personalize Runtime Runtime.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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();
```

Ottieni consigli con un filtro (gruppo di set di dati personalizzato).

```
// 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();
```

Ottieni consigli filtrati da un consulente creato in un gruppo di set di dati di 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();
```

- Per i dettagli sulle API, consulta la sezione API GetRecommendationsReference AWS SDK for JavaScript .

Esempi di Amazon Pinpoint con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Pinpoint.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Invia messaggi e-mail e di testo

Il seguente esempio di codice mostra come inviare e-mail e messaggi di testo con Amazon Pinpoint.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
//Set the MediaConvert Service Object
const pinClient = new PinpointClient({ region: REGION });
export { pinClient };
```

Invia un messaggio e-mail.

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

// The FromAddress must be verified in SES.
const fromAddress = "FROM_ADDRESS";
const toAddress = "TO_ADDRESS";
const projectId = "PINPOINT_PROJECT_ID";

// The subject line of the email.
var subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
```

```
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
 using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding for the subject line and message body of the email.
var charset = "UTF-8";

const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

const run = async () => {
```

```
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));

    const {
      MessageResponse: { Result },
    } = data;

    const recipientResult = Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    } else {
      console.log(recipientResult.MessageId);
    }
  } catch (err) {
    console.log(err.message);
  }
};

run();
```

Invia un messaggio SMS.

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

("use strict");

/* The phone number or short code to send the message from. The phone number
 or short code that you specify has to be associated with your Amazon Pinpoint
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number.  For best results, you should specify the phone
 number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
```

```
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderID";

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

const run = async () => {
```

```
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    return data; // For unit tests.
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  } catch (err) {
    console.log(err);
  }
};
run();
```

- Per i dettagli sull'API, consulta la SendMessagessezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invia un messaggio e-mail.

```
"use strict";

const AWS = require("aws-sdk");

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
```

```
var toAddress = "recipient@example.com";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https:\/\/aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
```

```
var params = {
  ApplicationId: appId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: senderAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

//Try to send the email.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
  } else {
    console.log(
      "Email sent! Message ID: ",
      data["MessageResponse"]["Result"][toAddress]["MessageId"]
    );
  }
});
```

Invia un messaggio SMS.

```
"use strict";

var AWS = require("aws-sdk");

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";

// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
var originationNumber = "+12065550199";

// The recipient's phone number.  For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
```

```
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderID";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

//Try to send the message.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
    // Otherwise, show the unique ID for the message.
  } else {
    console.log(
```

```
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  }
});
```

- Per i dettagli sull'API, consulta la SendMessagessezione AWS SDK for JavaScript API Reference.

Esempi di Amazon Redshift con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Redshift.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Creazione di un cluster

Il seguente esempio di codice mostra come creare un cluster Amazon Redshift.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Crea il cluster .

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

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

const run = async () => {
  try {
```

```
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, consulta la CreateClustersezione AWS SDK for JavaScript API Reference.

Eliminazione di un cluster

Il seguente esempio di codice mostra come eliminare un cluster Amazon Redshift.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Crea il cluster .

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

```
import { redshiftClient } from "./libs/redshiftClient.js";

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

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

- Per i dettagli sull'API, consulta la DeleteClustersezione AWS SDK for JavaScript API Reference.

Descrivi i tuoi cluster

Il seguente esempio di codice mostra come descrivere i cluster Amazon Redshift.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
```

```
export { redshiftClient };
```

Descrivi i tuoi cluster.

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

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

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

- Per i dettagli sull'API, consulta la sezione DescribeClusters AWS SDK for JavaScriptAPI Reference.

Modifica un cluster

Il seguente esempio di codice mostra come modificare un cluster Amazon Redshift.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Modifica un cluster.

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

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

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

- Per i dettagli sull'API, ModifyClusterconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon S3 con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon S3.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Amazon S3

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Amazon S3.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
};
```

- Per i dettagli sull'API, consulta la ListBucketssezione AWS SDK for JavaScript API Reference.

Argomenti

Azioni

Aggiunta di regole CORS a un bucket

Il seguente esempio di codice mostra come aggiungere regole CORS (Cross-Origin Resource Sharing) a un bucket S3.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiungi una regola 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutBucketCorssezione AWS SDK for JavaScript API Reference.

Aggiunta di una policy a un bucket

Il seguente esempio di codice mostra come aggiungere una policy a un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiungi la policy.

```
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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutBucketPolicysezione AWS SDK for JavaScript API Reference.

Copia di un oggetto da un bucket a un altro

Il seguente esempio di codice mostra come copiare un oggetto S3 da un bucket all'altro.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Copia l'oggetto.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la CopyObjectsezione AWS SDK for JavaScript API Reference.

Creazione di un bucket

Il seguente esempio di codice mostra come creare un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la CreateBucketsezione AWS SDK for JavaScript API Reference.

Eliminazione di una policy da un bucket

Il seguente esempio di codice mostra come eliminare una policy da un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina la policy 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteBucketPolicysezione AWS SDK for JavaScript API Reference.

Eliminazione di un bucket vuoto

Il seguente esempio di codice mostra come eliminare un bucket S3 vuoto.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina il 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteBucketsezione AWS SDK for JavaScript API Reference.

Eliminazione di un oggetto

Il seguente esempio di codice mostra come eliminare un oggetto S3.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina un oggetto.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la DeleteObjectsezione AWS SDK for JavaScript API Reference.

Eliminazione di più oggetti

Il seguente esempio di codice mostra come eliminare più oggetti da un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina più oggetti.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la DeleteObjectssezione AWS SDK for JavaScript API Reference.

Eliminazione della configurazione del sito Web da un bucket

Il seguente esempio di codice mostra come eliminare la configurazione del sito Web da un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina la configurazione del sito Web dal 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteBucketWebsitesezione AWS SDK for JavaScript API Reference.

Recupero di regole CORS per un bucket

Il seguente esempio di codice mostra come ottenere regole CORS (Cross-Origin Resource Sharing) per un bucket S3.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la policy CORS per il 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetBucketCorssezione AWS SDK for JavaScript API Reference.

Recupero di un oggetto da un bucket

Il seguente esempio di codice mostra come leggere i dati da un oggetto in un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Scarica l'oggetto.

```
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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetObjectsezione AWS SDK for JavaScript API Reference.

Recupero dell'ACL di un bucket

Il seguente esempio di codice mostra come ottenere l'elenco di controllo degli accessi (ACL) di un bucket S3.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera le autorizzazioni 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetBucketAclsezione AWS SDK for JavaScript API Reference.

Recupero della policy per un bucket

Il seguente esempio di codice mostra come ottenere la policy per un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la policy 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetBucketPolicysezione AWS SDK for JavaScript API Reference.

Recupero della configurazione del sito Web per un bucket

Il seguente esempio di codice mostra come ottenere la configurazione del sito Web per un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la configurazione del sito 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);
  }
};
```

- Per i dettagli sull'API, consulta la GetBucketWebsitesezione AWS SDK for JavaScript API Reference.

Elenco di bucket

Il seguente esempio di codice mostra come elencare i bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i bucket.

```
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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListBucketssezione AWS SDK for JavaScript API Reference.

Elenco di oggetti in un bucket

Il seguente esempio di codice mostra come elencare gli oggetti in un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca tutti gli oggetti nel bucket. Se è presente più di un oggetto, IsTruncated NextContinuationToken verrà utilizzato per scorrere l'elenco completo.

```
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);
  }
};
```

- Per i dettagli sull'API, consulta la versione ListObjectsV2 in AWS SDK for JavaScript API Reference.

Impostazione di una nuova ACL per un bucket

Il seguente esempio di codice mostra come impostare una nuova lista di controllo degli accessi (ACL) per un bucket S3.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inserisci l'ACL del bucket.

```
import {
  PutBucketAclCommand,
```

```
  GetBucketAclCommand,
  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);
```

```
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutBucketAclsezione AWS SDK for JavaScript API Reference.

Impostazione della configurazione del sito Web per un bucket

Il seguente esempio di codice mostra come impostare la configurazione del sito Web per un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Imposta la configurazione del sito 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutBucketWebsitesezione AWS SDK for JavaScript API Reference.

Caricamento di un oggetto in un bucket

Il seguente esempio di codice mostra come caricare un oggetto in un bucket S3.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Carica l'oggetto.

```
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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la PutObjectsezione AWS SDK for JavaScript API Reference.

Scenari

Creazione di un URL prefirmato

Il seguente esempio di codice mostra come creare un URL predefinito per Amazon S3 e caricare un oggetto.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un URL prefirmato per caricare un oggetto in 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);
  }
};
```

Crea un URL prefirmato per scaricare un oggetto da 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);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

Creare una pagina Web che elenca gli oggetti Amazon S3

Il codice di esempio seguente mostra come elencare gli oggetti Amazon S3 in una pagina Web.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Il codice seguente è il componente React pertinente che effettua chiamate all' AWS SDK. Una versione eseguibile dell'applicazione contenente questo componente è disponibile al link precedente. 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;
```

- Per i dettagli sull'API, consulta la sezione API ListObjectsReference AWS SDK for JavaScript .

Nozioni di base su bucket e oggetti

L'esempio di codice seguente mostra come:

- Crea un bucket e carica un file in tale bucket.
- Scaricare un oggetto da un bucket.
- Copiare un oggetto in una sottocartella in un bucket.
- Elencare gli oggetti in un bucket.
- Elimina il bucket e tutti gli oggetti in esso contenuti.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Innanzitutto, importa tutti i moduli necessari.

```
// 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";
```

Le importazioni precedenti fanno riferimento ad alcune utilità di supporto. Queste utilità sono locali al GitHub repository collegato all'inizio di questa sezione. Come riferimento, consulta le implementazioni di tali utilità riportate di seguito.

```
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`;
};
```

Gli oggetti in S3 sono archiviati in "bucket". Definiamo una funzione per creare un nuovo 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;
};
```

I bucket contengono "oggetti". Questa funzione carica il contenuto di una directory nel bucket come oggetti.

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

Dopo aver caricato gli oggetti, verifica che siano stati caricati correttamente. Puoi usare ListObjects per questo. Utilizzerai la proprietà 'Key', ma ci sono anche altre proprietà utili nella risposta.

```
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 volte potresti voler copiare un oggetto da un bucket ad altri bucket. Usa il CopyObject comando per questo.

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

Non esiste un metodo SDK per ottenere più oggetti da un bucket. Creerai invece un elenco di oggetti da scaricare ed eseguirai iterazioni su di essi.

```
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");
};
```

È il momento di eseguire una pulizia delle risorse. Prima di poter essere eliminato, un bucket deve essere vuoto. Queste due funzioni svuotano ed eliminano il 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 funzione "principale" esegue entrambe le operazioni. Se esegui direttamente questo file, verrà chiamata la funzione principale.

```
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);
  }
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2

- PutObject

Caricamento o download di file di grandi dimensioni

Il seguente esempio di codice mostra come caricare o scaricare file di grandi dimensioni da e verso Amazon S3.

Per ulteriori informazioni, consulta Caricamento di un oggetto utilizzando il caricamento in più parti.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Carica un file di grandi dimensioni.

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

Scarica un file di grandi dimensioni.

```
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);
};

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",
  });
};
```

Esempi di S3 Glacier che utilizzano SDK for (v3) JavaScript

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con S3 Glacier.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

Creazione di un vault

Il seguente esempio di codice mostra come creare un vault Amazon S3 Glacier.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
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 };
```

Crea il vault.

```
// 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();
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la CreateVaultsezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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!");
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la CreateVaultsezione AWS SDK for JavaScript API Reference.

Caricamento di un archivio su un vault

Il seguente esempio di codice mostra come caricare un archivio in un vault Amazon S3 Glacier.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
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 };
```

Caricamento dell'archivio.

```
// 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();
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la UploadArchivesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la UploadArchivesezione AWS SDK for JavaScript API Reference.

SageMaker esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. SageMaker

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve SageMaker

L'esempio di codice seguente mostra come iniziare a utilizzare SageMaker.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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;
};
```

- Per i dettagli sull'API, consulta la ListNotebookInstancessezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Crea una pipeline

Il seguente esempio di codice mostra come creare o aggiornare una pipeline in SageMaker.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Una funzione che crea una SageMaker pipeline utilizzando una definizione JSON fornita 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);

  const { PipelineArn } = await sagemakerClient.send(
    new CreatePipelineCommand({
      PipelineName: name,
      PipelineDefinition: pipelineDefinition,
      RoleArn: roleArn,
    }),
  );

  return {
    arn: PipelineArn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - CreatePipeline
  - UpdatePipeline

Elimina una pipeline

Il seguente esempio di codice mostra come eliminare una pipeline in. SageMaker

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

La sintassi per eliminare una SageMaker pipeline. Questo codice fa parte di una funzione più ampia. Fai riferimento a «Crea una pipeline» o al GitHub repository per ulteriori informazioni.

```
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
```

- Per i dettagli sull'API, consulta DeletePipelinela sezione API Reference.AWS SDK for JavaScript

Descrivi l'esecuzione di una pipeline

Il seguente esempio di codice mostra come descrivere l'esecuzione di una pipeline in. SageMaker

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Attendi che l'esecuzione di una SageMaker pipeline abbia successo, fallisca o si fermi.

```
/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
```

```
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient }) {
  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);
}
```

- Per i dettagli sull'API, consulta la sezione DescribePipelineExecution AWS SDK for JavaScriptAPI Reference.

Esegui una pipeline

Il seguente esempio di codice mostra come avviare l'esecuzione di una pipeline in. SageMaker

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Avvia l'esecuzione di una SageMaker pipeline.

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
```

```
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
```

```
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}
```

- Per i dettagli sull'API, consulta la sezione StartPipelineExecution AWS SDK for JavaScriptAPI Reference.

Scenari

Inizia con i lavori e le pipeline geospaziali

L'esempio di codice seguente mostra come:

- Imposta le risorse per una pipeline.
- Configura una pipeline che esegua un lavoro geospaziale.
- Avvio dell'esecuzione di una pipeline.
- Monitora lo stato dell'esecuzione.
- Visualizza l'output della pipeline.
- Pulisci le risorse.

Per ulteriori informazioni, consulta Creare ed eseguire SageMaker pipeline utilizzando gli AWS SDK su Community.aws.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Il seguente estratto di file contiene funzioni che utilizzano il SageMaker client per gestire una pipeline.

```
import { readFileSync } from "fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";
```

```
import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-
geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry, wait } 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 { Role } = await iamClient.send(
    new CreateRoleCommand({
      RoleName: name,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Action: ["sts:AssumeRole"],
            Principal: { Service: ["lambda.amazonaws.com"] },
          },
        ],
      }),
    }),
  );

  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 policy = {
    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 createPolicyCommand = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify(policy),
    PolicyName: name,
  });

  const { Policy } = await iamClient.send(createPolicyCommand);
  return {
    arn: Policy.Arn,
    policy,
    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`;

  const command = new CreateFunctionCommand({
    Code: {
      ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
    },
    Runtime: Runtime.nodejs18x,
    Handler: "index.handler",
    Layers: [layerVersionArn],
    FunctionName: name,
    Role: roleArn,
  });

  // 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 },
    () => lambdaClient.send(command),
  );

  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}} props
 */
export async function createSagemakerRole({ name, iamClient }) {
  const command = 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",
```

```
            ],
          },
        },
      ],
    }),
  });

  const { Role } = await iamClient.send(command);
  // Wait for the role to be ready.
  await wait(10);

  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 policy = {
    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 createPolicyCommand = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify(policy),
    PolicyName: name,
  });

  const { Policy } = await iamClient.send(createPolicyCommand);
  return {
    arn: Policy.Arn,
    policy,
    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);

  const { PipelineArn } = await sagemakerClient.send(
    new CreatePipelineCommand({
      PipelineName: name,
      PipelineDefinition: pipelineDefinition,
      RoleArn: roleArn,
    }),
  );

  return {
    arn: PipelineArn,
    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 { QueueUrl } = await sqsClient.send(
    new CreateQueueCommand({
      QueueName: name,
      Attributes: {
        DelaySeconds: "5",
        ReceiveMessageWaitTimeSeconds: "5",
        VisibilityTimeout: "300",
      },
```

```
    }),
  );

  const { Attributes } = await sqsClient.send(
    new GetQueueAttributesCommand({
      QueueUrl,
      AttributeNames: ["QueueArn"],
    }),
  );

  return {
    queueUrl: QueueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{lambdaName: string, queueArn: string, lambdaClient: import('@aws-sdk/
client-lambda').LambdaClient, sqsClient: import('@aws-sdk/client-sqs').SQSClient}}
 props
 */
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
}) {
  const { UUID } = await lambdaClient.send(
    new CreateEventSourceMappingCommand({
      EventSourceArn: queueArn,
      FunctionName: lambdaName,
    }),
  );

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
          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}} props
 */
export async function createS3Bucket({ name, s3Client }) {
  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}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient }) {
  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();
}
```

Questa funzione è un estratto da un file che utilizza le funzioni di libreria precedenti per configurare una SageMaker pipeline, eseguirla ed eliminare tutte le risorse create.

```
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 {
      // Run all of the clean up functions. If any fail, we log the error and
 continue.
      // This ensures all clean up functions are run.
      this.logger.logSeparator();
      const doCleanUp = await this.prompter.confirm({
        message: "Clean up resources?",
```

```
      });
      if (doCleanUp) {
        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,
    });
    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,
    });

    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"));
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution
  - UpdatePipeline

Esempi di Secrets Manager con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with Secrets Manager.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

Ottieni un valore segreto

I seguenti esempi di codice mostrano come recuperare un valore segreto di Gestione dei segreti.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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;
  }
};
```

- Per i dettagli sull'API, consulta la GetSecretValuesezione AWS SDK for JavaScript API Reference.

Esempi di Amazon SES con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon SES.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Creazione di un filtro di ricezione

L'esempio di codice seguente mostra come creare un filtro di ricezione di Amazon SES che blocca la posta in entrata da un indirizzo IP o un intervallo di indirizzi IP.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 (err) {
    console.log("Failed to create filter.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, consulta la CreateReceiptFiltersezione AWS SDK for JavaScript API Reference.

Creazione di una regola di ricezione

L'esempio di codice seguente mostra come creare una regola di ricezione di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
```

```
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
  });
};

const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};
```

- Per i dettagli sull'API, consulta la CreateReceiptRulesezione AWS SDK for JavaScript API Reference.

Creazione di un set di regole di ricezione

L'esempio di codice seguente mostra come creare una regola di ricezione di Amazon SES per organizzare le regole applicate alle e-mail in entrata.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la CreateReceiptRuleSetsezione AWS SDK for JavaScript API Reference.

Creazione di un modello di e-mail

L'esempio di codice seguente mostra come creare un modello di e-mail di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la CreateTemplatesezione AWS SDK for JavaScript API Reference.

Eliminazione di un filtro di ricezione

L'esempio di codice seguente mostra come eliminare un filtro di ricezione di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la DeleteReceiptFiltersezione AWS SDK for JavaScript API Reference.

Eliminazione di una regola di ricezione

L'esempio di codice seguente mostra come eliminare una regola di ricezione di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la DeleteReceiptRulesezione AWS SDK for JavaScript API Reference.

Eliminazione di un set di regole

L'esempio di codice seguente mostra come eliminare un set di regole di Amazon SES e tutte le regole in esso contenute.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
```

```
};
```

- Per i dettagli sull'API, consulta la DeleteReceiptRuleSetsezione AWS SDK for JavaScript API Reference.

Eliminazione di un modello di e-mail

L'esempio di codice seguente mostra come eliminare un modello di e-mail di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la DeleteTemplatesezione AWS SDK for JavaScript API Reference.

Eliminare un'identità

L'esempio di codice seguente mostra come eliminare un'identità di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la DeleteIdentitysezione AWS SDK for JavaScript API Reference.

Ottenimento di un modello di e-mail esistente

L'esempio di codice seguente mostra come ottenere un modello di e-mail di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 (err) {
    console.log("Failed to get email template.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, consulta la GetTemplatesezione AWS SDK for JavaScript API Reference.

Elenco di tutti i modelli di e-mail

L'esempio di codice seguente mostra come elencare tutti i modelli di e-mail di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la ListTemplatessezione AWS SDK for JavaScript API Reference.

Elenco di tutte le identità

Il seguente esempio di codice mostra come elencare le identità di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la ListIdentitiessezione AWS SDK for JavaScript API Reference.

Elenco di tutti i filtri di ricezione

L'esempio di codice seguente mostra come elencare tutti i filtri di ricezione di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
```

```
};
```

- Per i dettagli sull'API, consulta la ListReceiptFilterssezione AWS SDK for JavaScript API Reference.

Invio in blocco di un'e-mail basata su modello

L'esempio di codice seguente mostra come inviare a più destinazioni un'e-mail basata su modello con Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 (err) {
    console.log("Failed to send bulk template email", err);
    return err;
  }
};
```

- Per i dettagli sull'API, consulta la SendBulkTemplatedEmailsezione AWS SDK for JavaScript API Reference.

Invio di e-mail

Il seguente esempio di codice mostra come inviare e-mail con Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 (e) {
    console.error("Failed to send email.");
    return e;
  }
};
```

- Per i dettagli sull'API, consulta la SendEmailsezione AWS SDK for JavaScript API Reference.

Invio di e-mail non formattate

L'esempio di codice seguente mostra come inviare un'e-mail non formattata con Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Usare nodemailer per inviare un'e-mail con un allegato.

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

- Per i dettagli sull'API, consulta la SendRawEmailsezione AWS SDK for JavaScript API Reference.

Invio di un'e-mail basata su modello

L'esempio di codice seguente mostra come inviare un'e-mail basata su modello con Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 (err) {
    console.log("Failed to send template email", err);
    return err;
  }
};
```

- Per i dettagli sull'API, consulta la SendTemplatedEmailsezione AWS SDK for JavaScript API Reference.

Aggiornamento di un modello di e-mail

L'esempio di codice seguente mostra come aggiornare un modello di e-mail di Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la UpdateTemplatesezione AWS SDK for JavaScript API Reference.

Verificare un'identità di dominio

L'esempio di codice seguente mostra come verificare un'identità di dominio con Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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;
  }
};
```

- Per i dettagli sull'API, consulta la VerifyDomainIdentitysezione AWS SDK for JavaScript API Reference.

Verifica di un'identità e-mail

L'esempio di codice seguente mostra come verificare un'identità e-mail con Amazon SES.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Import 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;
  }
};
```

- Per i dettagli sull'API, consulta la VerifyEmailIdentitysezione AWS SDK for JavaScript API Reference.

Esempi di Amazon SNS con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon SNS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Amazon SNS

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inizializza un client SNS ed elenca gli argomenti nel tuo account.

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

- Per i dettagli sull'API, consulta la ListTopicssezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Verifica dell'esclusione di un numero di telefono

Il seguente esempio di codice mostra come verificare se un numero di telefono è disattivato dalla ricezione di messaggi Amazon SNS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la CheckIfPhoneNumberIsOptedOutsezione AWS SDK for JavaScript API Reference.

Conferma che un proprietario di endpoint desidera ricevere messaggi

Il seguente esempio di codice mostra come confermare che il proprietario di un endpoint desidera ricevere messaggi Amazon SNS convalidando il token inviato all'endpoint con una precedente azione Subscribe.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
```

```
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ConfirmSubscriptionsezione AWS SDK for JavaScript API Reference.

Creazione di un argomento

Il seguente esempio di codice mostra come creare un argomento Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la CreateTopicsezione AWS SDK for JavaScript API Reference.

Eliminazione di una sottoscrizione

Il seguente esempio di codice mostra come eliminare un abbonamento Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sulle API, consulta Annullamento della sottoscrizione nella Documentazione di riferimento per le API AWS SDK for JavaScript .

Eliminazione di un argomento

Il seguente esempio di codice mostra come eliminare un argomento di Amazon SNS e tutte le sottoscrizioni a tale argomento.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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
  //   }
  // }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la DeleteTopicsezione AWS SDK for JavaScript API Reference.

Come ottenere le proprietà di un argomento

Il seguente esempio di codice mostra come ottenere le proprietà di un argomento Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
```

```
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetTopicAttributessezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come configurarlo ed eseguirlo nel AWS Code Examples Repository.

Importare l'SDK e i moduli client e chiamare l'API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();

// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per i dettagli sull'API, consulta la GetTopicAttributessezione AWS SDK for JavaScript API Reference.

Come ottenere le impostazioni per l'invio di messaggi SMS

Il seguente esempio di codice mostra come ottenere le impostazioni per l'invio di messaggi SMS Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sulle API, consulta GetSMSAttributes nella Documentazione di riferimento per le API AWS SDK for JavaScript .

Come elencare i sottoscrittori di un argomento

Il seguente esempio di codice mostra come recuperare l'elenco degli abbonati di un argomento Amazon SNS.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListSubscriptionssezione AWS SDK for JavaScript API Reference.

Come elencare gli argomenti

Il seguente esempio di codice mostra come elencare gli argomenti di Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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;
```

```
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la ListTopicssezione AWS SDK for JavaScript API Reference.

Pubblicazione di un messaggio con un attributo

Il seguente esempio di codice mostra come pubblicare un messaggio con un attributo utilizzando Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Pubblica un messaggio in un argomento con opzioni di gruppo, duplicazione e attributo.

```
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();
    }
  }
```

- Per informazioni dettagliate sulle API, consulta Pubblicazione nella Documentazione di riferimento per le API AWS SDK for JavaScript .

Pubblicazione in un argomento

Il seguente esempio di codice mostra come pubblicare messaggi su un argomento di Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                  if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
```

```
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sulle API, consulta Pubblicazione nella Documentazione di riferimento per le API AWS SDK for JavaScript .

Configurazione delle impostazioni di default per l'invio di messaggi SMS

Il seguente esempio di codice mostra come configurare le impostazioni predefinite per l'invio di messaggi SMS tramite Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per informazioni dettagliate sulle API, consulta SetSMSAttributes nella Documentazione di riferimento per le API AWS SDK for JavaScript .

Impostazione degli attributi degli argomenti

Il seguente esempio di codice mostra come impostare gli attributi degli argomenti di Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la SetTopicAttributessezione AWS SDK for JavaScript API Reference.

Sottoscrizione di una funzione Lambda a un argomento

Il seguente esempio di codice mostra come sottoscrivere una funzione Lambda in modo che riceva notifiche da un argomento di Amazon SNS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sulle API, consulta Sottoscrizione nella Documentazione di riferimento sulle API AWS SDK for JavaScript .

Sottoscrizione di un'applicazione mobile a un argomento

Il seguente esempio di codice mostra come sottoscrivere un endpoint di applicazione mobile in modo che riceva notifiche da un argomento di Amazon SNS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sulle API, consulta Sottoscrizione nella Documentazione di riferimento sulle API AWS SDK for JavaScript .

Sottoscrizione di una coda SQS a un argomento

I seguenti esempi di codice mostrano come sottoscrivere una coda SQS in modo che riceva notifiche da un argomento Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
```

```
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- Per informazioni dettagliate sulle API, consulta Sottoscrizione nella Documentazione di riferimento sulle API AWS SDK for JavaScript .

Sottoscrizione di un indirizzo e-mail a un argomento

Il seguente esempio di codice mostra come iscrivere un indirizzo e-mail a un argomento di Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
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({});
```

Importare l'SDK e i moduli client e chiamare l'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'
```

```
  // }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sulle API, consulta Sottoscrizione nella Documentazione di riferimento sulle API AWS SDK for JavaScript .

Sottoscrizione con un filtro a un argomento

L'esempio di codice seguente mostra come sottoscrivere con un filtro a un argomento Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute set
 to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });
```

```
  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- Per informazioni dettagliate sulle API, consulta Sottoscrizione nella Documentazione di riferimento sulle API AWS SDK for JavaScript .

Scenari

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Creazione di un argomento (FIFO o non FIFO).
- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo è il punto di ingresso per questo flusso di lavoro.

```
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();
};
```

Il codice precedente fornisce le dipendenze necessarie e avvia il flusso di lavoro. La sezione successiva contiene il blocco principale dell'esempio.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes

- Pubblicare
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Esempi di Amazon SQS con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon SQS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve Amazon SQS

I seguenti esempi di codice mostrano come iniziare a usare Amazon SQS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inizializza un client Amazon SQS ed elenca le code.

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

- Per i dettagli sull'API, consulta la sezione API ListQueuesReference AWS SDK for JavaScript .

Argomenti

- Azioni
- Scenari

Azioni

Modifica la visibilità del timeout dei messaggi

Il seguente esempio di codice mostra come modificare la visibilità del timeout di un messaggio Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ricevi un messaggio Amazon SQS e modificane la visibilità del timeout.

```
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;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ChangeMessageVisibilityReference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ricevi un messaggio Amazon SQS e modificane la visibilità del timeout.

```
// 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");
    }
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ChangeMessageVisibilityReference.

Configura una coda di lettere non scritte

Il seguente esempio di codice mostra come configurare una coda di lettere non scritte in Amazon SQS.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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;
};
```

- Per i dettagli sull'API, consulta la SetQueueAttributessezione AWS SDK for JavaScript API Reference.

Crea una coda

Il seguente esempio di codice mostra come creare una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea una coda standard 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;
};
```

Crea una coda Amazon SQS con polling prolungato.

```
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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione API CreateQueueReference AWS SDK for JavaScript .

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea una coda standard 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);
  }
});
```

Crea una coda Amazon SQS che attende l'arrivo di un messaggio.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione API CreateQueueReference AWS SDK for JavaScript .

Elimina un batch di messaggi da una coda

Il seguente esempio di codice mostra come eliminare un batch di messaggi da una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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,
        })),
```

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

- Per i dettagli sull'API, consulta la DeleteMessageBatchsezione AWS SDK for JavaScript API Reference.

Eliminare un messaggio da una coda

Il seguente esempio di codice mostra come eliminare un messaggio da una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ricevi ed elimina messaggi 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,
        })),
      }),
    );
  }
};
```

- Per i dettagli sull'API, consulta la sezione DeleteMessage AWS SDK for JavaScriptAPI Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ricevi ed elimina messaggi 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);
      }
    });
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione DeleteMessage AWS SDK for JavaScriptAPI Reference.

Elimina una coda

Il seguente esempio di codice mostra come eliminare una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare una coda 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteQueueReference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare una coda 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteQueueReference.

Ottieni gli attributi per una coda

Il seguente esempio di codice mostra come ottenere gli attributi per una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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;
};
```

- Per i dettagli sull'API, consulta la GetQueueAttributessezione AWS SDK for JavaScript API Reference.

Ottieni l'URL di una coda

Il seguente esempio di codice mostra come ottenere l'URL di una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottieni l'URL per una coda 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetQueueUrlReference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottieni l'URL per una coda 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetQueueUrlReference.

Elencare code

Il seguente esempio di codice mostra come elencare le code Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le tue code 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListQueuesReference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le tue code 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListQueuesReference.

Ricevi messaggi da una coda

Il seguente esempio di codice mostra come ricevere messaggi da una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ricevi un messaggio da una coda 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,
        })),
      }),
    );
  }
};
```

Ricevi un messaggio da una coda Amazon SQS utilizzando il supporto per sondaggi lunghi.

```
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;
};
```

- Per i dettagli sulle API, consulta ReceiveMessagela sezione API Reference.AWS SDK for JavaScript

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ricevi un messaggio da una coda Amazon SQS utilizzando il supporto per sondaggi lunghi.

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sulle API, consulta ReceiveMessagela sezione API Reference.AWS SDK for JavaScript

Inviare un messaggio a una coda

Il seguente esempio di codice mostra come inviare un messaggio a una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invia un messaggio a una coda 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API SendMessageReference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invia un messaggio a una coda 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API SendMessageReference.

Imposta gli attributi della coda

Il seguente esempio di codice mostra come impostare gli attributi per una coda Amazon SQS.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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;
};
```

Configura una coda Amazon SQS per utilizzare il polling lungo.

```
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;
};
```

- Per i dettagli sull'API, consulta la sezione API SetQueueAttributesReference AWS SDK for JavaScript .

Scenari

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Creazione di un argomento (FIFO o non FIFO).
- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo è il punto di ingresso per questo flusso di lavoro.

```
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();
};
```

Il codice precedente fornisce le dipendenze necessarie e avvia il flusso di lavoro. La sezione successiva contiene il blocco principale dell'esempio.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Pubblicare
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Esempi di Step Functions con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with Step Functions.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Avvia una macchina a stati, esegui

Il seguente esempio di codice mostra come avviare l'esecuzione di una macchina a stati Step Functions.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
  return response;
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- Per i dettagli sull'API, consulta la StartExecutionsezione AWS SDK for JavaScript API Reference.

AWS STS esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. AWS STS

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Assunzione di un ruolo

Il seguente esempio di codice mostra come assumere un ruolo con AWS STS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
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 });
```

Assumi il ruolo 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);
  }
};
```

- Per i dettagli sull'API, consulta la AssumeRolesezione AWS SDK for JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

- Per i dettagli sull'API, consulta la AssumeRolesezione AWS SDK for JavaScript API Reference.

AWS Support esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. AWS Support

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve AWS Support

L'esempio di codice seguente mostra come iniziare a utilizzare AWS Support.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Richiama main() `per eseguire l'esempio.

```
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);
```

```
  }
};
```

- Per i dettagli sull'API, consulta la DescribeServicessezione AWS SDK for JavaScript API Reference.

Argomenti

- Azioni
- Scenari

Azioni

Come aggiungere una comunicazione a un caso

Il seguente esempio di codice mostra come aggiungere una AWS Support comunicazione con un allegato a una richiesta di supporto.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la AddCommunicationToCasesezione AWS SDK for JavaScript API Reference.

Come aggiungere un collegamento a un set

Il seguente esempio di codice mostra come aggiungere un AWS Support allegato a un set di allegati.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, consulta la AddAttachmentsToSetsezione AWS SDK for JavaScript API Reference.

Come creare un caso

Il seguente esempio di codice mostra come creare un nuovo AWS Support caso.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, CreateCaseconsulta AWS SDK for JavaScript API Reference.

Come descrivere un collegamento

Il seguente esempio di codice mostra come descrivere un allegato per un AWS Support caso.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, DescribeAttachmentconsulta AWS SDK for JavaScript API Reference.

Come descrivere casi

Il seguente esempio di codice mostra come descrivere i AWS Support casi.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, DescribeCasesconsulta AWS SDK for JavaScript API Reference.

Come descrivere comunicazioni

Il seguente esempio di codice mostra come descrivere AWS Support le comunicazioni relative a un caso.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, DescribeCommunicationsconsulta AWS SDK for JavaScript API Reference.

Come descrivere livelli di gravità

Il seguente esempio di codice mostra come descrivere i livelli di AWS Support gravità.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, DescribeSeverityLevelsconsulta AWS SDK for JavaScript API Reference.

Come risolvere un caso

Il seguente esempio di codice mostra come risolvere un AWS Support caso.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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);
  }
};
```

- Per i dettagli sull'API, ResolveCaseconsulta AWS SDK for JavaScript API Reference.

Scenari

Come iniziare con i casi

L'esempio di codice seguente mostra come:

- Ottieni e visualizza i servizi e i livelli di gravità disponibili per i casi.
- Crea una richiesta di supporto utilizzando un servizio, una categoria e un livello di gravità selezionato.
- Ottieni e visualizza un elenco di casi aperti per il giorno corrente.
- Aggiungi un set di collegamenti e una comunicazione al nuovo caso.
- Descrivi il nuovo collegamento e la nuova comunicazione per il caso.
- Risolvi il caso.
- Ottieni e visualizza un elenco di casi risolti per il giorno corrente.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo nel terminale.

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import inquirer from "inquirer";
```

```
// Retry an asynchronous function on failure.
const retry = async ({ intervalInMs = 500, maxRetries = 10 }, fn) => {
  try {
    return await fn();
  } catch (err) {
    console.log(`Function call failed. Retrying.`);
    console.error(err.message);
    if (maxRetries === 0) throw err;
    await new Promise((resolve) => setTimeout(resolve, intervalInMs));
    return retry({ intervalInMs, maxRetries: maxRetries - 1 }, fn);
  }
};

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;
    }
  }
};

// Get the list of available services.
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const { selectedService } = await inquirer.prompt({
    name: "selectedService",
    type: "list",
    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;
};

// Get the list of available support case categories for a service.
export const getCategory = async (service) => {
  const { selectedCategory } = await inquirer.prompt({
    name: "selectedCategory",
    type: "list",
    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.prompt({
    name: "selectedSeverityLevel",
    type: "list",
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

// Create a new support case and return the caseId.
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;
};

// Get an attachment set.
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.prompt({
    name: "shouldResolve",
    type: "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.
export const findCase = async ({ caseId, cases, nextToken }) => {
  const foundCase = cases.find((c) => c.caseId === caseId);

  if (foundCase) {
    return foundCase;
  }

  if (nextToken) {
    const response = await client.send(
      new DescribeCasesCommand({
        nextToken,
        includeResolvedCases: true,
      })
    );
    return findCase({
      caseId,
      cases: response.cases,
      nextToken: response.nextToken,
    });
  }

  throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};
```

```
const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
 select one.
    const selectedCategory = await getCategory(selectedService);

    // Provide the severity available severity levels for the account and prompt the
 user to select one.
    const selectedSeverityLevel = await getSeverityLevel();

    // Create a support case.
    console.log("\nCreating a support case.");
    caseId = await createCase({
      selectedService,
      selectedCategory,
      selectedSeverityLevel,
    });
    console.log(`Support case created: ${caseId}`);

    // Display a list of open support cases created today.
    const todaysOpenCases = await retry(
      { intervalInMs: 1000, maxRetries: 15 },
      getTodaysOpenCases
    );
    console.log(
      `\nOpen support cases created today: ${todaysOpenCases.length}`
    );
    console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

    // Create an attachment set.
    console.log("\nCreating an attachment set.");
    const attachmentSetId = await createAttachmentSet();
    console.log(`Attachment set created: ${attachmentSetId}`);

    // Add the attachment set to the support case.
```

```
    console.log(`\nAdding attachment set to ${caseId}`);
    await linkAttachmentSetToCase(attachmentSetId, caseId);
    console.log(`Attachment set added to ${caseId}`);

    // List the communications for a support case.
    console.log(`\nListing communications for ${caseId}`);
    const communications = await getCommunications(caseId);
    console.log(
      communications
        .map(
          (c) =>
            `Communication created on ${c.timeCreated}. Has
${c.attachmentSet.length} attachments.`
        )
        .join("\n")
    );

    // Describe the first attachment.
    console.log(`\nDescribing attachment ${attachmentSetId}`);
    const attachmentId = getFirstAttachment(communications);
    const attachment = await getAttachment(attachmentId);
    console.log(
      `Attachment is the file '${
        attachment.fileName
      }' with data: \n${new TextDecoder().decode(attachment.data)}`
    );

    // Confirm that the support case should be resolved.
    const isResolved = await resolveCase(caseId);
    if (isResolved) {
      // List the resolved cases and include the one previously created.
      // Resolved cases can take a while to appear.
      console.log(
        "\nWaiting for case status to be marked as resolved. This can take some
time."
      );
      const resolvedCases = await retry(
        { intervalInMs: 20000, maxRetries: 15 },
        () => getTodaysResolvedCases(caseId)
      );
      console.log("Resolved cases:");
      console.log(resolvedCases.map((c) => c.caseId).join("\n"));
    }
  } catch (err) {
```

```
    console.error(err);
  }
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for JavaScript .
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases
  - DescribeCommunications
  - DescribeServices
  - DescribeSeverityLevels
  - ResolveCase

Esempi di Amazon Transcribe con SDK JavaScript for (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Transcribe.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

Eliminare un processo di trascrizione medica

Il seguente esempio di codice mostra come eliminare un processo di trascrizione di Amazon Transcribe Medical.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
const { TranscribeClient } = require("@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 };
```

Eliminare un processo di trascrizione medica.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
```

```
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteMedicalTranscriptionJobconsulta AWS SDK for JavaScript API Reference.

Eliminare un processo di trascrizione

Il seguente esempio di codice mostra come eliminare un processo di trascrizione di Amazon Transcribe.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare un processo di trascrizione.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
```

```
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Crea il client.

```
const { TranscribeClient } = require("@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 };
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteTranscriptionJobconsulta AWS SDK for JavaScript API Reference.

Elencare i processi di trascrizione medica

Il seguente esempio di codice mostra come elencare i lavori di trascrizione di Amazon Transcribe Medical.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
const { TranscribeClient } = require("@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 };
```

Elencare i processi di trascrizione medica.

```
// 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();
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListMedicalTranscriptionJobsconsulta AWS SDK for JavaScript API Reference.

Elencare i processi di trascrizione

Il seguente esempio di codice mostra come elencare i lavori di trascrizione di Amazon Transcribe.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elencare i processi di trascrizione.

```
// 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();
```

Crea il client.

```
const { TranscribeClient } = require("@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 };
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListTranscriptionJobsconsulta AWS SDK for JavaScript API Reference.

Avviare un processo di trascrizione medica

Il seguente esempio di codice mostra come avviare un processo di trascrizione di Amazon Transcribe Medical.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
const { TranscribeClient } = require("@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 };
```

Avviare un processo di trascrizione medica.

```
// 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();
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, StartMedicalTranscriptionJobconsulta AWS SDK for JavaScript API Reference.

Avviare un processo di trascrizione

Il seguente esempio di codice mostra come avviare un processo di trascrizione di Amazon Transcribe.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Avviare un processo di trascrizione.

```
// 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();
```

Crea il client.

```
const { TranscribeClient } = require("@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 };
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, StartTranscriptionJobconsulta AWS SDK for JavaScript API Reference.

Esempi interservizi che utilizzano SDK for JavaScript (v3)

Le seguenti applicazioni di esempio utilizzano AWS SDK for JavaScript (v3) per funzionare su più applicazioni. Servizi AWS

Gli esempi trasversali mirano a un livello avanzato di esperienza per aiutarti a iniziare a creare applicazioni.

Esempi

- Creazione di un'app Amazon Transcribe
- Creazione di un'app in streaming Amazon Transcribe
- Costruisci un'applicazione per inviare dati a una tabella DynamoDB
- Crea un chatbot Amazon Lex per coinvolgere i visitatori del tuo sito web
- Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette
- Creazione di un'applicazione Web per tracciare i dati DynamoDB
- Creazione di un tracciatore di elementi di lavoro di Aurora Serverless
- Creazione di un'applicazione Amazon Textract explorer
- Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio
- Rileva i DPI nelle immagini con Amazon Rekognition utilizzando un SDK AWS

- Rileva oggetti nelle immagini con Amazon Rekognition utilizzando un SDK AWS
- Rileva persone e oggetti in un video con Amazon Rekognition utilizzando un SDK AWS
- Richiamo a una funzione Lambda da un browser
- Utilizzo di un'API Gateway per richiamare una funzione Lambda
- Utilizzo di Step Functions per richiamare le funzioni Lambda
- Utilizzo degli eventi pianificati per richiamare una funzione Lambda

Creazione di un'app Amazon Transcribe

SDK per JavaScript (v3)

Crea un'app che utilizza Amazon Transcribe per trascrivere e visualizzare le registrazioni vocali nel browser. L'app utilizza due bucket Amazon Simple Storage Service (Amazon S3), uno per ospitare il codice dell'applicazione e l'altro per archiviare le trascrizioni. L'app utilizza un pool di utenti Amazon Cognito per autenticare gli utenti. Gli utenti autenticati dispongono delle autorizzazioni AWS Identity and Access Management (IAM) per accedere ai servizi richiesti. AWS

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3 .

Servizi utilizzati in questo esempio

- Amazon Cognito Identity
- Amazon S3
- Amazon Transcribe

Creazione di un'app in streaming Amazon Transcribe

SDK per JavaScript (v3)

Mostra come utilizzare Amazon Transcribe per creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia i risultati per e-mail tramite Amazon Simple Email Service (Amazon SES).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Costruisci un'applicazione per inviare dati a una tabella DynamoDB

SDK per JavaScript (v3)

Questo esempio mostra come creare un'app che consenta agli utenti di inviare dati a una tabella Amazon DynamoDB e un messaggio di testo all'amministratore utilizzando Amazon Simple Notification Service (Amazon SNS).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3 .

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SNS

Crea un chatbot Amazon Lex per coinvolgere i visitatori del tuo sito web

SDK per JavaScript (v3)

Mostra come utilizzare l'API Amazon Lex per creare un Chatbot all'interno di un'applicazione Web per coinvolgere i visitatori del sito Web.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo Costruire un chatbot Amazon Lex nella guida per gli AWS SDK for JavaScript sviluppatori.

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette

SDK per JavaScript (v3)

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Creazione di un'applicazione Web per tracciare i dati DynamoDB

SDK per JavaScript (v3)

Mostra come utilizzare l'API Amazon DynamoDB per creare un'applicazione Web dinamica che traccia i dati di lavoro DynamoDB.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SES

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

SDK per JavaScript (v3)

Mostra come utilizzare AWS SDK for JavaScript (v3) per creare un'applicazione Web che tenga traccia degli elementi di lavoro in un database Amazon Aurora e invii report tramite e-mail utilizzando Amazon Simple Email Service (Amazon SES). Questo esempio utilizza un front-end creato con React.js per interagire con un backend Express Node.js.

- Integra un'applicazione web React.js con. Servizi AWS
- Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
- Invia un report per e-mail degli elementi di lavoro filtrati tramite Amazon SES.
- Distribuisci e gestisci risorse di esempio con lo AWS CloudFormation script incluso.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Aurora
- Amazon RDS
- Servizi di dati di Amazon RDS
- Amazon SES

Creazione di un'applicazione Amazon Textract explorer

SDK per JavaScript (v3)

Mostra come utilizzare per AWS SDK for JavaScript creare un'applicazione React che utilizza Amazon Textract per estrarre dati dall'immagine di un documento e visualizzarli in una pagina Web interattiva. Questo esempio viene eseguito in un browser Web e richiede, come credenziali, un'identità autenticata Amazon Cognito. Utilizza Amazon Simple Storage Service (Amazon S3) per l'archiviazione e per le notifiche esegue il polling di una coda di Servizio di coda semplice

Amazon (Amazon SQS) sottoscritta a un argomento Servizio di notifica semplice Amazon (Amazon SNS).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio

SDK per JavaScript (v3)

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in. GitHub I seguenti estratti mostrano come AWS SDK for JavaScript viene utilizzato all'interno delle funzioni Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";
```

```
/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
```

```
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
```

```
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });
```

```
  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Rileva i DPI nelle immagini con Amazon Rekognition utilizzando un SDK AWS

SDK per (v3) JavaScript

Mostra come usare Amazon Rekognition AWS SDK for JavaScript con la per creare un'applicazione per rilevare i dispositivi di protezione individuale (DPI) nelle immagini che si trovano in un bucket Amazon Simple Storage Service (Amazon S3). L'applicazione salva i risultati in una tabella Amazon DynamoDB e invia all'amministratore una notifica e-mail sui risultati tramite Amazon Simple Email Service (Amazon SES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizzare le immagini per rilevare i DPI tramite Amazon Rekognition.
- Verificare un indirizzo e-mail per Amazon SES.
- Aggiornare una tabella DynamoDB con i risultati.
- Inviare una notifica e-mail tramite Amazon SES.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- DynamoDB

- Amazon Rekognition
- Amazon S3
- Amazon SES

Rileva oggetti nelle immagini con Amazon Rekognition utilizzando un SDK AWS

SDK per (v3) JavaScript

Mostra come usare Amazon Rekognition AWS SDK for JavaScript con la per creare un'app che utilizzi Amazon Rekognition per identificare gli oggetti per categoria nelle immagini che si trovano in un bucket Amazon Simple Storage Service (Amazon S3). L'applicazione invia all'amministratore una notifica e-mail sui risultati tramite Amazon Simple Email Service (Amazon SES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per rilevare gli oggetti tramite Amazon Rekognition.
- Verificare un indirizzo e-mail per Amazon SES.
- Inviare una notifica e-mail tramite Amazon SES.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Amazon S3
- Amazon SES

Rileva persone e oggetti in un video con Amazon Rekognition utilizzando un SDK AWS

SDK per (v3) JavaScript

Mostra come usare Amazon Rekognition AWS SDK for JavaScript con la per creare un'app per rilevare volti e oggetti nei video che si trovano in un bucket Amazon Simple Storage Service

(Amazon S3). L'applicazione invia all'amministratore una notifica e-mail sui risultati tramite Amazon Simple Email Service (Amazon SES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizzare le immagini per rilevare i DPI tramite Amazon Rekognition.
- Verificare un indirizzo e-mail per Amazon SES.
- Inviare una notifica e-mail tramite Amazon SES.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Amazon S3
- Amazon SES

Richiamo a una funzione Lambda da un browser

SDK per JavaScript (v2)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

SDK per JavaScript (v3)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente. Questa app utilizza la versione 3. AWS SDK for JavaScript

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

Utilizzo di un'API Gateway per richiamare una funzione Lambda

SDK per JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando l'API di JavaScript runtime Lambda. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda richiamata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3 .

Servizi utilizzati in questo esempio

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Utilizzo di Step Functions per richiamare le funzioni Lambda

SDK per JavaScript (v3)

Mostra come creare un flusso di lavoro AWS serverless utilizzando AWS Step Functions and. AWS SDK for JavaScript Ogni fase del flusso di lavoro viene implementata utilizzando una AWS Lambda funzione.

Lambda è un servizio di calcolo che consente di eseguire il codice senza effettuare il provisioning o la gestione di server. Step Functions è un servizio di orchestrazione serverless che consente di combinare funzioni Lambda e altri servizi AWS per la creazione di applicazioni business-critical.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3 .

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Utilizzo degli eventi pianificati per richiamare una funzione Lambda

SDK per JavaScript (v3)

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3 .

Servizi utilizzati in questo esempio

- DynamoDB
- EventBridge
- Lambda

- Amazon SNS

Sicurezza per questo AWS prodotto o servizio

La sicurezza cloud di Amazon Web Services (AWS) è la priorità più alta. In quanto cliente AWS , è possibile trarre vantaggio da un'architettura di data center e di rete progettata per soddisfare i requisiti delle organizzazioni più esigenti a livello di sicurezza. La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud.

Security of the Cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce tutti i servizi offerti nel AWS Cloud e della fornitura di servizi che è possibile utilizzare in modo sicuro. La nostra responsabilità in AWS materia di sicurezza è la massima priorità e l'efficacia della nostra sicurezza viene regolarmente testata e verificata da revisori di terze parti nell'ambito dei Programmi di AWS conformità.

Sicurezza nel cloud: la responsabilità dell'utente è determinata dal AWS servizio utilizzato e da altri fattori, tra cui la sensibilità dei dati, i requisiti dell'organizzazione e le leggi e i regolamenti applicabili.

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Argomenti

Protezione dei dati in questo AWS prodotto o servizio

Il modello di responsabilità AWS condivisa di si applica alla protezione dei dati in questo AWS prodotto o servizio. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. Inoltre, sei responsabile della configurazione della

protezione e delle attività di gestione per i Servizi AWS che utilizzi. Per ulteriori informazioni sulla privacy dei dati, vedi Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog AWS Shared Responsibility Model and GDPR nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-2 per l'accesso AWS tramite un'interfaccia a riga di comando o un'API, utilizza un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-2.

Ti consigliamo vivamente di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori con questo AWS prodotto o servizio o altro Servizi AWS utilizzando la console, l'API o gli SDK. AWS CLI AWS I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Identity and Access Management

AWS Identity and Access Management (IAM) è un software Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle autorizzazioni) a utilizzare le risorse. AWS IAM è uno Servizio AWS strumento che puoi utilizzare senza costi aggiuntivi.

Argomenti

Destinatari

Il modo in cui usi AWS Identity and Access Management (IAM) varia a seconda del lavoro che AWS svolgi.

Utente del servizio: se lo utilizzi Servizi AWS per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più AWS funzionalità per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso ti consente di richiedere le autorizzazioni corrette all'amministratore. Se non riesci ad accedere a una funzionalità di AWS, consulta Risoluzione dei problemi di AWS identità e accesso o consulta la guida per l'utente della funzionalità Servizio AWS che stai utilizzando.

Amministratore del servizio: se sei responsabile delle AWS risorse della tua azienda, probabilmente hai pieno accesso a AWS. È tuo compito determinare a quali AWS funzionalità e risorse devono accedere gli utenti del servizio. Devi inviare le richieste all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per saperne di più su come la tua azienda può utilizzare IAM con AWS, consulta la guida per l'utente del Servizio AWS software che stai utilizzando.

Amministratore IAM: un amministratore IAM potrebbe essere interessato a ottenere dei dettagli su come scrivere policy per gestire l'accesso a AWS. Per visualizzare esempi di policy AWS basate sull'identità che puoi utilizzare in IAM, consulta la guida per l'utente di quella Servizio AWS che stai utilizzando.

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi essere autenticato (aver effettuato l' Utente root dell'account AWS accesso AWS) come utente IAM o assumendo un ruolo IAM.

Puoi accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAM Identity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS accesso. Per ulteriori informazioni sull'accesso a AWS, vedi Come accedere al tuo Account AWS nella Guida per l'Accedi ad AWS utente.

Se accedi a AWS livello di codice, AWS fornisce un kit di sviluppo software (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le tue richieste utilizzando le tue credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sull'utilizzo del metodo consigliato per firmare autonomamente le richieste, consulta Signing AWS API request nella IAM User Guide.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta Autenticazione a più fattori nella Guida per l'utente di AWS IAM Identity Center e Utilizzo dell'autenticazione a più fattori (MFA) in AWS nella Guida per l'utente di IAM.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte Servizi AWS le risorse dell'account. Questa identità è denominata utente Account AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzarle per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione Attività che richiedono le credenziali dell'utente root nella Guida per l'utente di IAM.

Identità federata

Come procedura consigliata, richiedi agli utenti umani, compresi gli utenti che richiedono l'accesso come amministratore, di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente dell'elenco utenti aziendale, di un provider di identità Web AWS Directory Service, della directory Identity Center o di qualsiasi utente che accede utilizzando le Servizi AWS credenziali fornite tramite un'origine di identità. Quando le identità federate accedono Account AWS, assumono ruoli e i ruoli forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. Puoi creare utenti e gruppi in IAM Identity Center oppure puoi connetterti e sincronizzarti con un set di utenti e gruppi nella tua fonte di identità per utilizzarli su tutte le tue applicazioni. Account AWS Per ulteriori informazioni sul Centro identità IAM, consulta Cos'è Centro identità IAM? nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un utente IAM è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, per casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine nella Guida per l'utente di IAM.

Un gruppo IAM è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, è possibile avere un gruppo denominato Amministratori IAM e concedere a tale gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta Quando creare un utente IAM (invece di un ruolo) nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Puoi assumere temporaneamente un ruolo IAM in AWS Management Console cambiando ruolo. Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per

ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta Utilizzo di ruoli IAM nella Guida per l'utente di IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- Accesso utente federato: per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta Creazione di un ruolo per un provider di identità di terza parte nella Guida per l'utente di IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per ulteriori informazioni sui set di autorizzazioni, consulta Set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
- Autorizzazioni utente IAM temporanee: un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- Accesso multi-account: è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente di IAM.
- Accesso a più servizi: alcuni Servizi AWS utilizzano le funzionalità di altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è comune che tale servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.
  - Sessioni di accesso diretto (FAS): quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra azione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama an Servizio AWS, combinate con la richiesta Servizio AWS per effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le operazioni. Per i dettagli delle policy relative alle richieste FAS, consulta la pagina Forward access sessions.

  - Ruolo di servizio: un ruolo di servizio è un ruolo IAM assunto da un servizio per eseguire operazioni per conto dell'utente. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS nella Guida per l'utente di IAM.
  - Ruolo collegato al servizio: un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.
- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un'istanza EC2 e che AWS CLI effettuano richieste API. AWS Ciò è preferibile all'archiviazione delle chiavi di accesso nell'istanza EC2. Per assegnare un AWS ruolo a un'istanza EC2 e renderlo disponibile per tutte le sue applicazioni, crei un profilo di istanza collegato all'istanza. Un profilo dell'istanza contiene il ruolo e consente ai programmi in esecuzione sull'istanza EC2 di ottenere le credenziali temporanee. Per ulteriori informazioni, consulta Utilizzo di un ruolo IAM per concedere autorizzazioni ad applicazioni in esecuzione su istanze di Amazon EC2 nella Guida per l'utente di IAM.

Per informazioni sull'utilizzo dei ruoli IAM, consulta Quando creare un ruolo IAM (invece di un utente) nella Guida per l'utente di IAM.

Gestione dell'accesso con policy

Puoi controllare l'accesso AWS creando policy e collegandole a AWS identità o risorse. Una policy è un oggetto AWS che, se associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste politiche quando un principale (utente, utente root o sessione di ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per ulteriori informazioni sulla struttura e sui contenuti dei documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente di IAM.

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire azioni sulle risorse di cui hanno bisogno, un amministratore IAM

può creare policy IAM. Successivamente l'amministratore può aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'operazione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'azione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dall' AWS Management Console AWS CLI, dall'o dall' AWS API.

Policy basate su identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Creazione di policy IAM nella Guida per l'utente di IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono incorporate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli nel tuo Account AWS. Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scelta fra policy gestite e policy inline nella Guida per l'utente di IAM.

Policy basate su risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile allegare a una risorsa. Gli esempi più comuni di policy basate su risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarle per controllare l'accesso a una risorsa specifica. Quando è allegata a una risorsa, una policy definisce le azioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non puoi utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Liste di controllo degli accessi (ACL)

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni per accedere a una risorsa. Le ACL sono simili alle policy basate su risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano gli ACL. AWS WAF Per maggiori informazioni sulle ACL, consulta Panoramica delle liste di controllo degli accessi (ACL) nella Guida per gli sviluppatori di Amazon Simple Storage Service.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi e meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- Limiti delle autorizzazioni: un limite delle autorizzazioni è una funzione avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo IAM). È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal` sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni sui limiti delle autorizzazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente di IAM.
- Politiche di controllo dei servizi (SCP): le SCP sono politiche JSON che specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa (OU) in. AWS Organizations AWS Organizations è un servizio per il raggruppamento e la gestione centralizzata di più Account AWS di proprietà dell'azienda. Se abiliti tutte le funzionalità in un'organizzazione, puoi applicare le policy di controllo dei servizi (SCP) a uno o tutti i tuoi account. L'SCP limita le autorizzazioni per le entità negli account dei membri, inclusa ciascuna. Utente root dell'account AWS Per ulteriori informazioni su organizzazioni e policy SCP, consulta la pagina sulle Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations .
- Policy di sessione: le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni, consulta Policy di sessione nella Guida per l'utente di IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella IAM User Guide.

Come Servizi AWS lavorare con IAM

Per avere una visione di alto livello di come Servizi AWS funziona la maggior parte delle funzionalità IAM, consulta AWS i servizi che funzionano con IAM nella IAM User Guide.

Per scoprire come utilizzare uno specifico Servizio AWS con IAM, consulta la sezione sulla sicurezza della Guida per l'utente del servizio pertinente.

Risoluzione dei problemi di AWS identità e accesso

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con un AWS IAM.

Argomenti

- Non sono autorizzato a eseguire alcuna azione in AWS
- Non sono autorizzato a eseguire iam: PassRole
- Voglio consentire a persone esterne a me di accedere Account AWS alle mie AWS risorse

Non sono autorizzato a eseguire alcuna azione in AWS

Se ricevi un errore che indica che non sei autorizzato a eseguire un'operazione, le tue policy devono essere aggiornate per poter eseguire l'operazione.

L'errore di esempio seguente si verifica quando l'utente IAM `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa *`my-example-widget`* fittizia ma non dispone di autorizzazioni `awes:`*`GetWidget`* fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In questo caso, la policy per l'utente `mateojackson` deve essere aggiornata per consentire l'accesso alla risorsa *`my-example-widget`* utilizzando l'azione `awes:`*`GetWidget`*.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un errore che indica che non sei autorizzato a eseguire l'operazione `iam:PassRole`, le tue policy devono essere aggiornate per poter passare un ruolo a AWS.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

L'errore di esempio seguente si verifica quando un utente IAM denominato `marymajor` cerca di utilizzare la console per eseguire un'operazione in AWS. Tuttavia, l'operazione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per passare il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie AWS risorse

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per servizi che supportano policy basate su risorse o liste di controllo accessi (ACL), utilizza tali policy per concedere alle persone l'accesso alle tue risorse.

Per ulteriori informazioni, consulta gli argomenti seguenti:

- Per sapere se AWS supporta queste funzionalità, consultaCome Servizi AWS lavorare con IAM.
- Per scoprire come fornire l'accesso alle tue risorse attraverso Account AWS le risorse di tua proprietà, consulta Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà nella IAM User Guide.

- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta Fornire l'accesso a soggetti Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso a utenti autenticati esternamente (Federazione delle identità) nella Guida per l'utente di IAM.
- Per informazioni sulle differenze tra l'utilizzo di ruoli e policy basate su risorse per l'accesso multi-account, consulta Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente IAM.

Convalida della conformità per questo AWS prodotto o servizio

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione Scope by Compliance Program Servizi AWS e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di AWS conformità Programmi di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in AWS Artifact .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWS fornisce le seguenti risorse per contribuire alla conformità:

- Guide introduttive su sicurezza e conformità: queste guide all'implementazione illustrano considerazioni sull'architettura e forniscono passaggi per implementare ambienti di base incentrati sulla AWS sicurezza e la conformità.
- Progettazione per la sicurezza e la conformità HIPAA su Amazon Web Services: questo white paper descrive in che modo le aziende possono utilizzare AWS per creare applicazioni idonee all'HIPAA.

> **Note**
>
> Non Servizi AWS tutte sono idonee all'HIPAA. Per ulteriori informazioni, consulta la sezione Riferimenti sui servizi conformi ai requisiti HIPAA.

- AWS Risorse per la per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe essere valida per il tuo settore e la tua località.

- AWS Guide alla conformità dei clienti: comprendi il modello di responsabilità condivisa attraverso la lente della conformità. Le guide riassumono le migliori pratiche per la protezione Servizi AWS e mappano le linee guida per i controlli di sicurezza su più framework (tra cui il National Institute of Standards and Technology (NIST), il Payment Card Industry Security Standards Council (PCI) e l'International Organization for Standardization (ISO)).
- Valutazione delle risorse con regole nella Guida per gli AWS Config sviluppatori: il AWS Config servizio valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida e alle normative del settore.
- AWS Security Hub— Ciò Servizio AWS fornisce una visione completa dello stato di sicurezza interno. AWS La Centrale di sicurezza utilizza i controlli di sicurezza per valutare le risorse AWS e verificare la conformità agli standard e alle best practice del settore della sicurezza. Per un elenco dei servizi e dei controlli supportati, consulta la pagina Documentazione di riferimento sui controlli della Centrale di sicurezza.
- AWS Audit Manager— Ciò Servizio AWS consente di verificare continuamente AWS l'utilizzo per semplificare la gestione dei rischi e la conformità alle normative e agli standard di settore.

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Resilienza per questo AWS prodotto o servizio

L'infrastruttura AWS globale è costruita attorno a zone Regioni AWS di disponibilità.

Regioni AWS forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti.

Con le zone di disponibilità, puoi progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, vedere Global Infrastructure.AWS

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS

servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Sicurezza dell'infrastruttura per questo AWS prodotto o servizio

Questo AWS prodotto o servizio utilizza servizi gestiti ed è pertanto protetto dalla sicurezza di rete AWS globale. Per informazioni sui servizi AWS di sicurezza e su come AWS protegge l'infrastruttura, consulta AWS Cloud Security. Per progettare il tuo AWS ambiente utilizzando le migliori pratiche per la sicurezza dell'infrastruttura, vedi Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Utilizzate chiamate API AWS pubblicate per accedere a questo AWS Prodotto o Servizio attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. O puoi utilizzare AWS Security Token Service (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Applica una versione TLS minima

Per aumentare la sicurezza durante la comunicazione con AWS i servizi, configurali AWS SDK for JavaScript per utilizzare TLS 1.2 o versioni successive.

> **Important**
>
> La versione AWS SDK for JavaScript v3 negozia automaticamente la versione TLS di più alto livello supportata da un determinato endpoint del servizio. AWS Facoltativamente, puoi

imporre una versione TLS minima richiesta dall'applicazione, ad esempio TLS 1.2 o 1.3, ma tieni presente che TLS 1.3 non è supportato da alcuni endpoint del AWS servizio, quindi alcune chiamate potrebbero non riuscire se applichi TLS 1.3.

Transport Layer Security (TLS) è un protocollo utilizzato dai browser Web e da altre applicazioni per garantire la privacy e l'integrità dei dati scambiati su una rete.

Verificare e applicare TLS in Node.js

Quando si utilizza AWS SDK for JavaScript con Node.js, il livello di sicurezza Node.js sottostante viene utilizzato per impostare la versione TLS.

Node.js 12.0.0 e versioni successive utilizzano una versione minima di OpenSSL 1.1.1b, che supporta TLS 1.3. La versione AWS SDK for JavaScript 3 utilizza per impostazione predefinita TLS 1.3 quando disponibile, ma utilizza per impostazione predefinita una versione precedente se necessario.

Verificare la versione di OpenSSL e TLS

Per ottenere la versione di OpenSSL utilizzata da Node.js sul computer, eseguire il seguente comando.

```
node -p process.versions
```

La versione di OpenSSL nell'elenco è la versione utilizzata da Node.js, come mostrato nell'esempio seguente.

```
openssl: '1.1.1b'
```

Per ottenere la versione di TLS utilizzata da Node.js sul computer, avviare la shell Node ed eseguire i seguenti comandi, in ordine.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

L'ultimo comando restituisce la versione TLS, come mostrato nell'esempio seguente.

```
'TLSv1.3'
```

Node.js utilizza per impostazione predefinita questa versione di TLS e tenta di negoziare un'altra versione di TLS se una chiamata non ha esito positivo.

Applicazione di una versione minima di TLS

Node.js negozia una versione di TLS quando una chiamata non riesce. È possibile applicare la versione TLS minima consentita durante questa negoziazione, sia quando si esegue uno script dalla riga di comando che per richiesta nel codice. JavaScript

Per specificare la versione TLS minima dalla riga di comando, è necessario utilizzare Node.js versione 11.0.0 o successiva. Per installare una versione specifica di Node.js, installa innanzitutto Node Version Manager (nvm) utilizzando i passaggi disponibili in Installazione e aggiornamento del gestore delle versioni di Node. Quindi eseguire i seguenti comandi per installare e utilizzare una versione specifica di Node.js.

```
nvm install 11
nvm use 11
```

Enforce TLS 1.2

Per stabilire che TLS 1.2 sia la versione minima consentita, specificare l'argomento `--tls-min-v1.2` durante l'esecuzione dello script, come mostrato nell'esempio seguente.

```
node --tls-min-v1.2 yourScript.js
```

Per specificare la versione TLS minima consentita per una richiesta specifica nel JavaScript codice, utilizzate il `httpOptions` parametro per specificare il protocollo, come illustrato nell'esempio seguente.

```
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

Per far sì che TLS 1.3 sia la versione minima consentita, specificate l'`--tls-min-v1.3`argomento durante l'esecuzione dello script, come illustrato nell'esempio seguente.

```
node --tls-min-v1.3 yourScript.js
```

Per specificare la versione TLS minima consentita per una richiesta specifica nel JavaScript codice, utilizzate il `httpOptions` parametro per specificare il protocollo, come illustrato nell'esempio seguente.

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

Verificare e applicare TLS in uno script del browser

Quando utilizzate l'SDK per JavaScript in uno script del browser, le impostazioni del browser controllano la versione di TLS utilizzata. La versione di TLS utilizzata dal browser non può essere individuata o impostata dallo script e deve essere configurata dall'utente. Per verificare e applicare la versione di TLS utilizzata in uno script del browser, consultare le istruzioni relative al browser specifico.

Microsoft Internet Explorer

1. Apri Internet Explorer.
2. Dalla barra dei menu, scegli Strumenti - Opzioni Internet - scheda Avanzate.
3. Scorri verso il basso fino alla categoria Sicurezza, seleziona manualmente la casella di opzione Usa TLS 1.2.
4. Fai clic su OK.
5. Chiudi il browser e riavvia Internet Explorer.

Microsoft Edge

1. Nella casella di ricerca del menu di Windows, digita *`Opzioni Internet`*.
2. In Best match, fai clic su Opzioni Internet.
3. Nella finestra Proprietà Internet, nella scheda Avanzate, scorri verso il basso fino alla sezione Sicurezza.
4. Seleziona la casella di controllo User TLS 1.2.
5. Fai clic su OK.

Google Chrome

1. Apri Google Chrome.
2. Fai clic su Alt F e seleziona Impostazioni.
3. Scorri verso il basso e seleziona Mostra impostazioni avanzate... .
4. Scorri verso il basso fino alla sezione Sistema e fai clic su Apri impostazioni proxy... .
5. Seleziona la scheda Avanzate.
6. Scorri verso il basso fino alla categoria Sicurezza, seleziona manualmente la casella di opzione Usa TLS 1.2.
7. Fai clic su OK.
8. Chiudi il browser e riavvia Google Chrome.

Mozilla Firefox

1. Apri Firefox.
2. Nella barra degli indirizzi, digita about:config e premi Invio.

3. Nel campo Cerca, inserisci tls. Trova e fai doppio clic sulla voce security.tls.version.min.
4. Imposta il valore intero su 3 per forzare il protocollo TLS 1.2 a diventare quello predefinito.
5. Fai clic su OK.
6. Chiudi il browser e riavvia Mozilla Firefox.

Apple Safari

Non ci sono opzioni per abilitare i protocolli SSL. Se utilizzi la versione 7 o successiva di Safari, TLS 1.2 viene abilitato automaticamente.

Migrazione alla versione 3

La sezione spiega come migrare dalla versione 2 alla versione 3 di. AWS SDK for JavaScript

Migra il codice su SDK per V3 JavaScript

AWS SDK for JavaScript la versione 3 (v3) include interfacce modernizzate per le configurazioni e le utilità dei client, che includono credenziali, caricamento multiparte di Amazon S3, client di documenti DynamoDB, camerieri e altro ancora. Puoi trovare cosa è cambiato nella v2 e nelle versioni equivalenti della v3 per ogni modifica nella guida alla migrazione sul repository. AWS SDK for JavaScript GitHub

Per sfruttare appieno la AWS SDK for JavaScript v3, consigliamo di utilizzare gli script codemod descritti di seguito.

Usa codemod per migrare il codice v2 esistente

La raccolta di script codemod in aws-sdk-js-codemodaiuta a migrare l'applicazione esistente AWS SDK for JavaScript (v2) per utilizzare le API v3. È possibile eseguire la trasformazione come segue.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

Ad esempio, considera di avere il codice seguente, che crea un client Amazon DynamoDB dalla v2 e chiama l'operazione. `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);
});
```

Puoi eseguire la nostra `v2-to-v3` trasformazione nel modo seguente. `example.ts`

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

La trasformazione convertirà l'importazione di DynamoDB in v3, creerà un client v3 e chiamerà l'operazione come segue. `listTables`

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

Abbiamo implementato trasformazioni per casi d'uso comuni. Se il codice non si trasforma correttamente, crea una segnalazione di bug o una richiesta di funzionalità con codice di input di esempio e codice di output osservato/previsto. Se il tuo caso d'uso specifico è già stato segnalato in un problema esistente, mostra il tuo supporto con un voto positivo.

Cronologia dei documenti per AWS SDK for JavaScript la versione 3

Cronologia dei documenti

La tabella seguente descrive le modifiche importanti nella versione V3 del AWS SDK for JavaScript20 ottobre 2020 in poi. Per ricevere notifiche sugli aggiornamenti di questa documentazione, puoi sottoscrivere un feed RSS.

Modifica	Descrizione	Data
Annuncio	Banner superiore aggiornato con un end-of-support promemoria per Internet Explorer 11.	23 settembre 2022
Aggiornamenti minori	Aggiornamenti minori alla chiarezza e alla risoluzione dei link interrotti. Sono stati aggiunti link di sensibilizzazione agli AWS SDK e alla Guida di riferimento agli strumenti.	22 agosto 2022
Applica una versione TLS minima	Sono state aggiunte informazioni su TLS 1.3.	31 marzo 2022
Tutorial aggiornato AWS Lambda	È stato aggiunto un tutorial che dimostra come creare un'applicazione basata su browser per l'invio di dati a una tabella Amazon DynamoDB.	20 ottobre 2020
Impostazione delle credenziali nell'argomento Node.js aggiornato	Aggiorna l'argomento sull'impostazione delle credenziali in	20 ottobre 2020

	Node.js per AWS SDK for JavaScript V3.	
Migrare alla V3	È stato aggiunto un argomento per descrivere come migrare alla V3. AWS SDK for JavaScript	20 ottobre 2020
Guida introduttiva	Argomenti aggiornati per iniziare a usare il browser e usare Node.js per AWS SDK for JavaScript V3.	20 ottobre 2020
Generatore di browser	Le informazioni su AWS Browser Builder sono state rimosse perché non sono necessarie per AWS SDK for JavaScript V3.	20 ottobre 2020
Esempi di servizi Amazon Transcribe aggiornati	Esempi di servizi Amazon Transcribe aggiornati per V3. AWS SDK for JavaScript	20 ottobre 2020
Esempi di servizi Amazon Simple Notification Service aggiornati	Esempi di servizi Amazon Simple Notification Service aggiornati per la AWS SDK for JavaScript versione 3.	20 ottobre 2020
Esempi di servizi Amazon Simple Email Service aggiornati	Esempi di servizi Amazon Simple Email Service aggiornati per la AWS SDK for JavaScript versione 3.	20 ottobre 2020
Esempi di servizi Amazon Redshift aggiornati	Esempi di servizi Amazon Redshift aggiornati per AWS SDK for JavaScript V3.	20 ottobre 2020

Esempi di servizi Amazon Lex aggiornati	Esempi di servizi Amazon Lex aggiornati per la AWS SDK for JavaScript versione 3.	20 ottobre 2020
Esempi di servizi Amazon DynamoDB aggiornati	Esempi di servizi Amazon DynamoDB aggiornati per V3. AWS SDK for JavaScript	20 ottobre 2020
AWS Elemental MediaConvert esempi di servizi aggiornati	Esempi AWS Elemental MediaConvert di servizi aggiornati per AWS SDK for JavaScript V3.	20 ottobre 2020
AWS Lambda esempi di servizi aggiornati	Esempi AWS Lambda di servizi aggiornati per AWS SDK for JavaScript V3.	20 ottobre 2020
AWS SDK for JavaScript Anteprima della V3 Developer Guide	Rilasciata la versione preliminare della AWS SDK for JavaScript V3 Developer Guide.	19 ottobre 2020