Guida per gli sviluppatori per la SDK versione 3

AWS SDK for JavaScript

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

Table of Contents

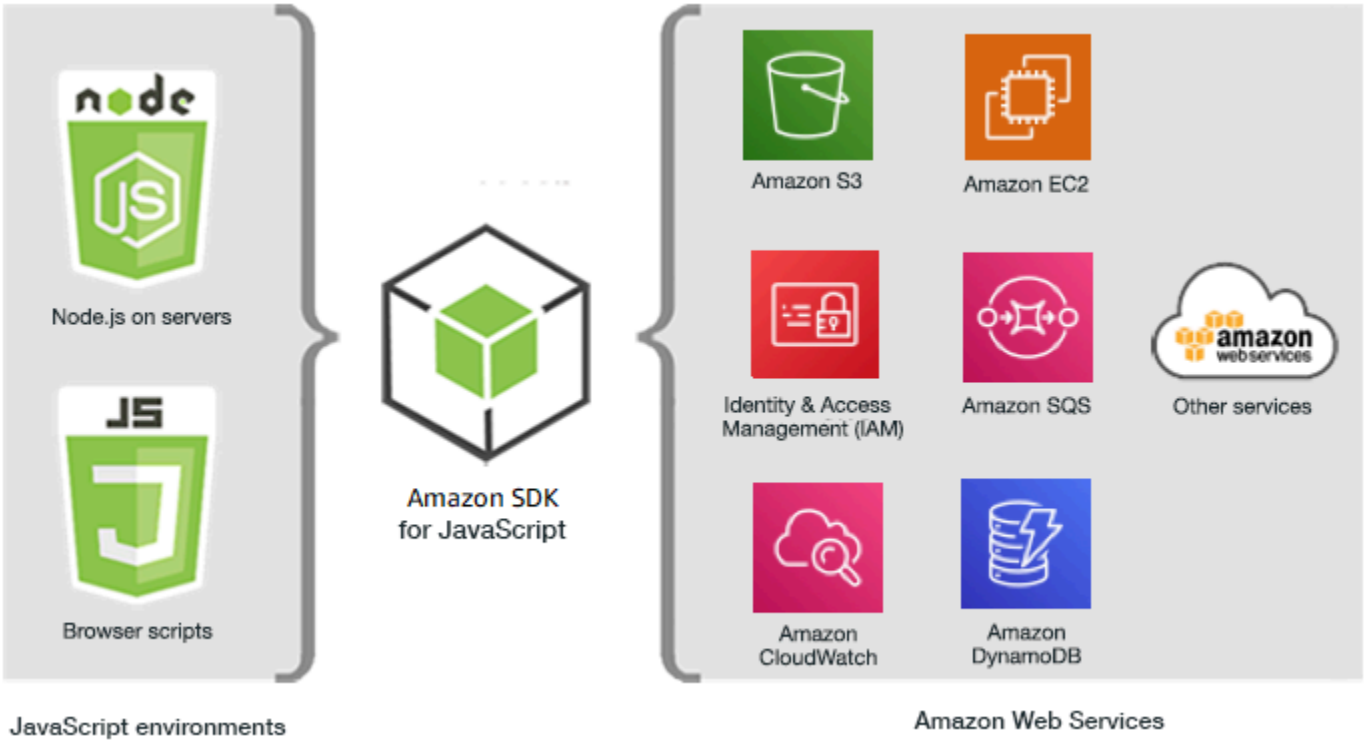
La guida API di riferimento AWS SDK for JavaScript V3 descrive in dettaglio tutte le API operazioni per la AWS SDK for JavaScript versione 3 (V3).

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 il AWS SDK for JavaScript?

Benvenuto nella Guida per gli AWS SDK for JavaScript sviluppatori. Questa guida fornisce informazioni generali sulla configurazione e la AWS SDK for JavaScript configurazione di. Inoltre, illustra esempi e tutorial sull' AWS SDK for JavaScript esecuzione 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.



Inizia a usare l'SDK

Se sei pronto a provare l'SDK, segui gli esempi all'indirizzo. Inizia a usare

Per configurare il tuo ambiente di sviluppo, consulta. Configura l'SDK per JavaScript

Se attualmente utilizzi la versione 2.x di SDK per JavaScript, consulta Migrare alla v3 per indicazioni specifiche.

Se stai cercando esempi di codice per, consulta. AWS serviziSDKper JavaScript esempi di codice (v3)

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:

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

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 usare Node.js per scrivere funzioni su richiesta AWS Lambda .

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 Cloud9 IDE. 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**
>
> 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 da AWS SDK for JavaScript, vedere. 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>` Per 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.
- Utilizzalo AWS Lambda per incapsulare una logica proprietaria che puoi richiamare dagli script del browser senza scaricare e rivelare la tua proprietà intellettuale agli utenti.

Informazioni sugli esempi

Puoi cercare JavaScript esempi nell'SDK nel AWS Code Example Repository.

Risorse

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

- AWS SDK for JavaScript Guida di riferimento all'API V3
- AWS Guida 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
- AWS Repository di esempi di codice
- Canale Gitter
- Stack Overflow
- Domande Stack Overflow etichettate AWS -sdk-js
- GitHub
  - Fonte SDK
  - Fonte della documentazione

Inizia con AWS SDK for JavaScript

AWS SDK for JavaScript Fornisce 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, vedi. Utilizzare AWS Cloud9 con AWS SDK for JavaScript

Argomenti

Autenticazione SDK con AWS

È necessario stabilire in che modo il codice si autentica AWS durante lo sviluppo con. AWS serviziÈ possibile configurare l'accesso programmatico alle AWS risorse in diversi modi a seconda dell'ambiente e dell' AWS accesso a disposizione.

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

- Il AWS CLI, che viene utilizzato per avviare una sessione del portale di AWS accesso 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 AWS servizi 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 `sso-session` sezione contiene le impostazioni per avviare una sessione del portale di AWS accesso.

```
[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).

Avviare una sessione del portale di accesso AWS

Prima di eseguire un'applicazione che consente l'accesso AWS servizi, 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 AWS accesso, esegui il seguente comando in. AWS CLI

```
aws sso login
```

Se hai seguito le istruzioni e disponi di una configurazione predefinita del profilo, non è necessario richiamare il comando con un'`--profile`opzione. 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 AWS accesso ed eseguia`ws sso login`, non ti verrà richiesto di fornire credenziali.
> La procedura di accesso potrebbe richiedere all'utente di consentire l' AWS CLI accesso ai dati. 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 agli AWS ambienti e alle applicazioni dell'utente. Gli utenti umani che fanno parte della tua organizzazione, ovvero tu, lo sviluppatore, sono noti come identità della forza lavoro.

Utilizza credenziali temporanee per l'accesso. AWS Puoi utilizzare un provider di identità per i tuoi utenti umani per fornire l'accesso federato agli AWS account 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 AWS credenziali a breve termine, consulta Temporary Security Credentials nella IAM User Guide.
- Per ulteriori informazioni su altri provider di credenziali AWS SDK for JavaScript V3, consulta Fornitori di credenziali standardizzati nella Guida di riferimento agli SDK e agli AWS 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 SDKper JavaScript esempi di 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, get StartedRole) nel nome del ruolo IAM.
6. In Configure properties, inserisci un nome (ad esempio get StartedPool) nel nome del pool di identità.

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, get). StartedPool 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 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, get StartedRole), 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 ReadOnlyAmazonS3 Access.

> **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. Accedi AWS Management Console e apri 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 AWS Code 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, get). StartedRole
- Nella console Amazon Cognito, elimina il nome del pool di identità (ad esempio, get StartedPool).

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'EC2istanza Amazon.
- 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";
```

Configura il SDK per JavaScript

Prima di utilizzare il modulo SDK per JavaScript richiamare i servizi Web utilizzando ilAPI, è necessario configurare il. SDK Come minimo, è necessario 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 AWS SDKsand Tools Reference Guide contiene anche impostazioni, funzionalità e altri concetti fondamentali comuni a molti di essi. AWS SDKs

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

Argomenti

Configurazione per servizio

È possibile configurarla SDK passando le informazioni di configurazione a un oggetto di servizio.

La configurazione a livello di servizio fornisce 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 globale. SDK `AWS.config` Inoltre, richiama `AWS.config.update({/* params *})` solo la configurazione aggiornata per i client di servizio istanziata 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 da. 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

A ogni servizio utilizzato nel modulo SDK JavaScript si accede tramite un oggetto di servizio che fa parte del API 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 a EC2 oggetti Amazon in più AWS regioni, crea un oggetto di EC2 servizio Amazon 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 AWS regione

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). Si specifica una regione quando si crea un client di servizio nel modulo SDK in JavaScript modo che SDK acceda al servizio in quella regione. Alcuni servizi sono disponibili solo in regioni specifiche.

SDKPer impostazione predefinita, il for JavaScript non seleziona una regione. Tuttavia, è possibile 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, 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 da utilizzare da parte diSDK, è possibile conservare la AWS regione e altre impostazioni di configurazione in un file condiviso denominato appositamente `config` per l'SDKuso. Se la variabile di `AWS_SDK_LOAD_CONFIG` ambiente è impostata su un valore vero, il comando SDK for cerca JavaScript automaticamente un `config` file al momento del caricamento. 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 strumenti AWS SDKs 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. API Le credenziali possono essere impostate per servizio, passando le credenziali direttamente a un oggetto 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 privilegio minimo dell'argomento Best Practice nella Guida per l'utente. IAM

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. AWS servizio 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 = 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'utilizzo AWS SDK for JavaScript di Node.js può differire dall'utilizzo degli script SDK nel browser. Alcuni API metodi funzionano in Node.js ma non negli script del browser, e viceversa. E il corretto utilizzo di alcuni di APIs essi dipende dalla familiarità con i modelli di codifica più comuni di Node.js, come l'importazione e l'utilizzo di altri moduli Node.js come il modulo. `File System (fs)`

Usa 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 HTTP modulo integrato, utilizzare quanto segue.

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

Richiama i metodi del modulo come se fossero metodi di quell'oggetto. Ad esempio, ecco un codice che legge un HTML file.

```
// 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.

Configura maxSockets in Node.js

Su Node.js, è possibile impostare il numero massimo di connessioni per origine. Se `maxSockets` è impostata, il HTTP client 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
  });
});
```

Il SDK for JavaScript utilizza il `maxSockets` valore 50 se non si fornisce un valore o un `Agent` oggetto. Se fornite un `Agent` oggetto, verrà utilizzato `maxSockets` il suo valore. Per ulteriori informazioni sull'impostazione `maxSockets` in Node.js, consultate la documentazione di 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'HTTPSagente Node.jsHTTP/predefinito crea una nuova TCP connessione per ogni nuova richiesta. Per evitare il costo di stabilire una nuova connessione, AWS SDK for JavaScript riutilizza le TCP connessioni per impostazione predefinita.

Per operazioni di breve durata, come le query Amazon DynamoDB, il sovraccarico di latenza dovuto alla TCP configurazione di una connessione 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.

Se non si desidera riutilizzare TCP le connessioni, è possibile disattivare il riutilizzo di queste connessioni in tempo reale per client per servizio, come illustrato nell'esempio seguente per un client DynamoDB. `keepAlive`

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

const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpsAgent: new Agent({ keepAlive: false })
    })
});
```

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

Configura i proxy per Node.js

Se non riesci a connetterti direttamente a Internet, il modulo JavaScript supporta SDK l'uso dei HTTPS proxy HTTP tramite un agente di terze partiHTTP.

Per trovare un HTTP agente di terze parti, cerca "HTTPproxy» su npm.

Per installare un HTTP agente proxy 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 AWS SDK for JavaScript browser.

Argomenti

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

Crea il file per i browser SDK

SDKA differenza della JavaScript versione 2 (V2), V3 non viene fornito come JavaScript file con supporto incluso per un set predefinito di servizi. V3 consente invece di raggruppare e includere nel browser solo JavaScript i file necessari, riducendo SDK il sovraccarico. Ti consigliamo di utilizzare Webpack SDK per raggruppare JavaScript i file necessari 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'SDKesterno di un ambiente che viene applicato CORS nel tuo browser e desideri accedere a tutti i servizi forniti dal SDK for JavaScript, puoi creare una copia personalizzata di SDK localmente clonando il repository ed eseguendo gli stessi strumenti di compilazione che creano la versione ospitata predefinita di. SDK Le sezioni seguenti descrivono i passaggi per crearlo SDK con servizi e versioni aggiuntivi. API

Usa il SDK Builder per creare il SDK 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:

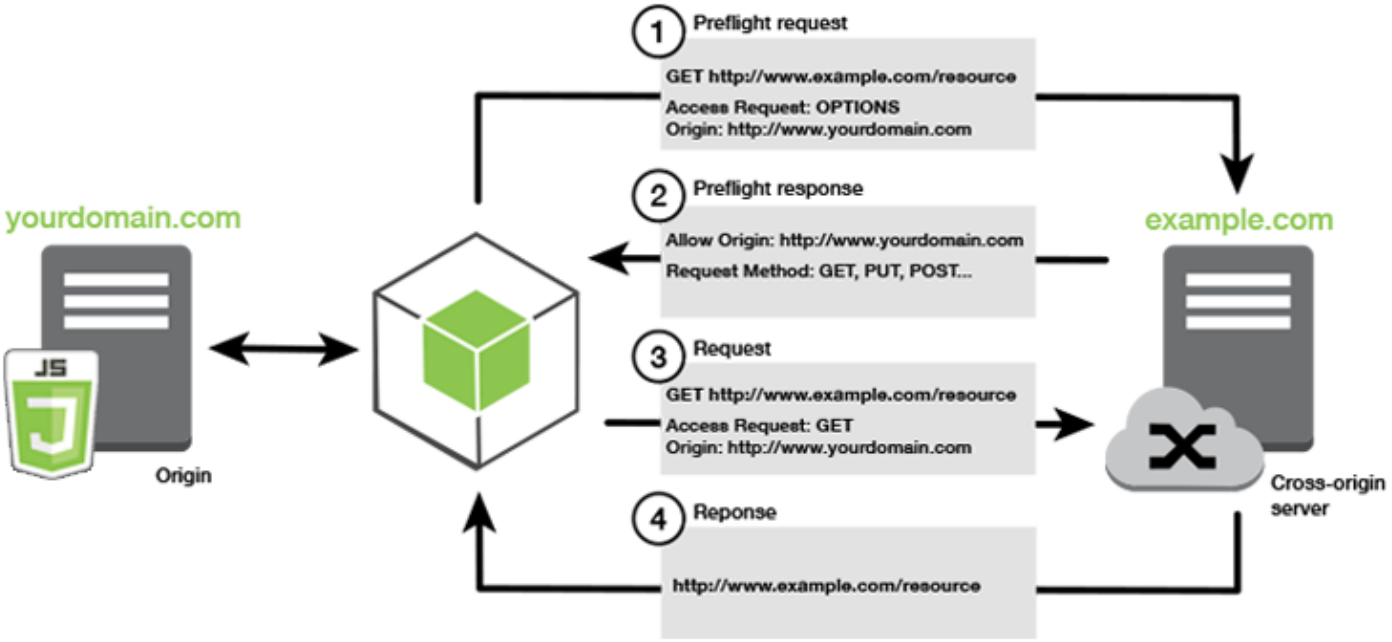
- Il dominio specifico che effettua la richiesta
- Il tipo di richiesta HTTP effettuata (GET, PUT, POST, DELETE e così via)

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 SDK disponibili JavaScript

La AWS SDK for JavaScript v3 fornisce l'accesso ai servizi che supporta attraverso 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 include una o più classi di client che offrono funzionalità e risorse APIs di servizio di basso livello. Ad esempio, Amazon APIs DynamoDB è disponibile tramite la classe. `DynamoDB`

I servizi esposti tramite il form JavaScript seguono lo schema di richiesta-risposta SDK per scambiare messaggi con le applicazioni chiamanti. In questo schema, il codice che richiama un servizio invia una HTTPS richiestaHTTP/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 di un AWS servizio include l'intero ciclo di vita della richiesta e della 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

JavaScript APISupporta la maggior parte dei AWS servizi disponibili. Ogni servizio di JavaScript API fornisce una classe client con un `send` metodo da utilizzare per richiamare tutti i supporti API del servizio. Per ulteriori informazioni sulle classi di servizio, le operazioni e i parametri di JavaScript API, vedere la Guida APIdi riferimento.

Quando si utilizza il file SDK in Node.js, si aggiunge il SDK pacchetto per ogni servizio necessario all'applicazione`import`, che fornisce il 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 chiamate un metodo di un oggetto di servizio, passate i parametri JSON come richiesto daAPI. 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 passaggio dei JSON parametri, consulta. Utilizzo delle JSON

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

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

Chiama i servizi in modo asincrono

Tutte le richieste effettuate tramite sono asincroneSDK. 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

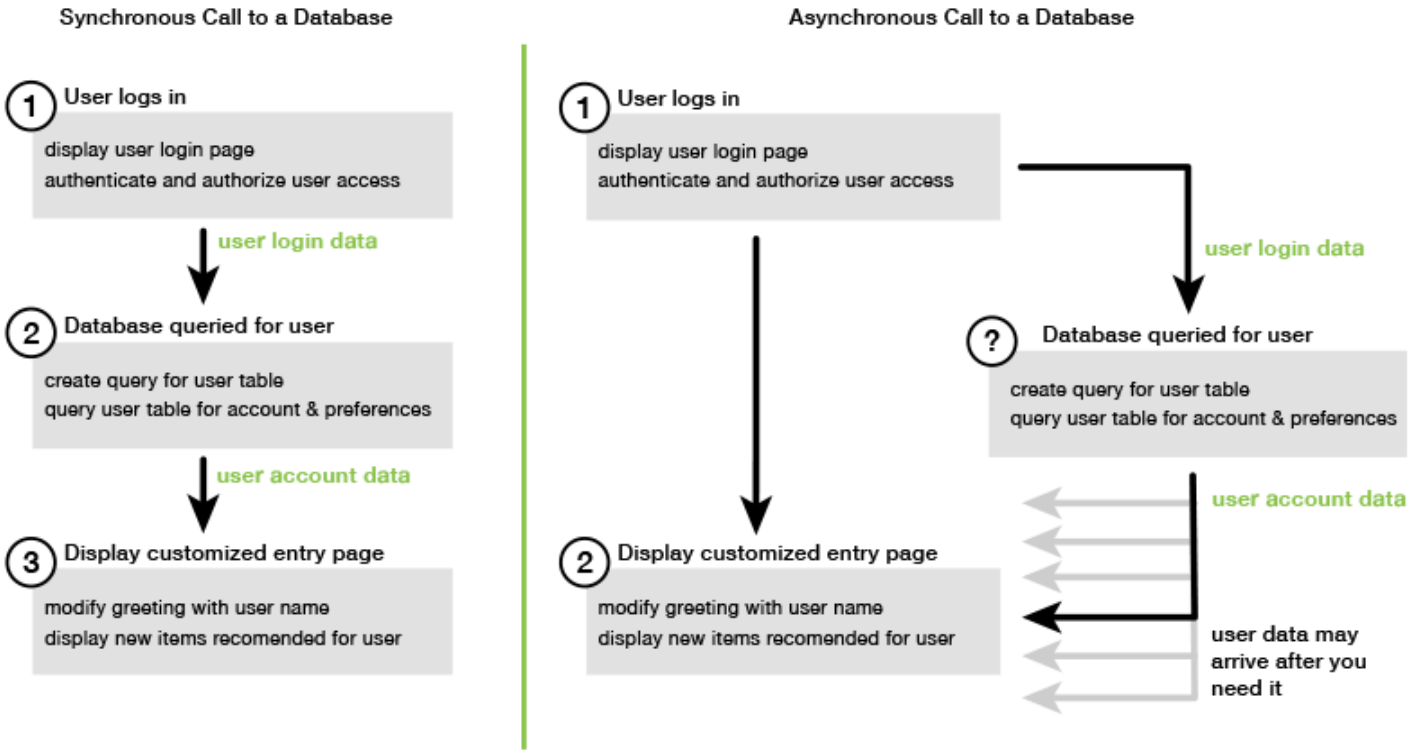
Gestisci le chiamate asincrone

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. `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 JavaScript le promesse native (ECMAScript2015) 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 ECMAScript compatibilità 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) del SDK for JavaScript consente di inviare richieste.

> **Note**
>
> È inoltre possibile eseguire operazioni utilizzando i comandi della versione 2 (V2) quando si utilizza la V3 del for. SDK 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 JSON oggetto di richiesta con i valori per la richiesta, ad esempio il nome di un bucket Amazon S3 specifico. Puoi esaminare i parametri della richiesta consultando l'argomento di API riferimento relativo all'interfaccia con il nome associato al metodo client. Ad esempio, se si utilizza il *`AbcCommand`* metodo 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 *`AbcCommand`* metodo client, l'oggetto di risposta è di *`AbcResponse`* tipo (interfaccia).

Dati di accesso 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

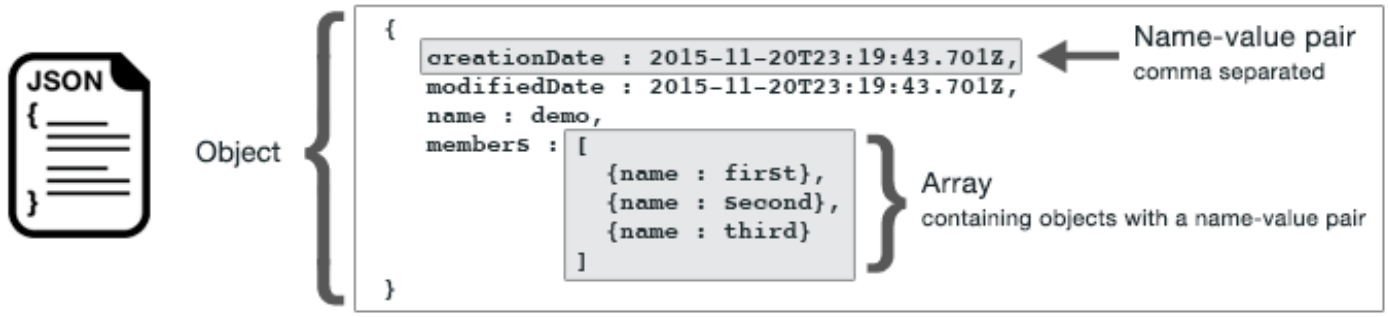
Se un comando fallisce, genera un'eccezione. Il seguente frammento di codice mostra un modo per gestire un'eccezione di servizio.

```
try {
  await client.send(someCommand);
} catch (e) {
  if (e.name === "InvalidSignatureException") {
    // Handle InvalidSignatureException
  } else if (e.name === "ResourceNotFoundException") {
    // Handle ResourceNotFoundException
  } else if (e.name === "FooServiceException") {
    // Handle all other server-side exceptions from Foo service
  } else {
    // Handle errors from SDK
  }
}
```

Utilizzo delle JSON

JSONè 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 è indipendente da qualsiasi linguaggio di programmazione. JSON

Viene AWS SDK for JavaScript utilizzato JSON per inviare dati a oggetti di servizio quando si effettuano richieste e riceve dati da oggetti di servizio as. JSON Per ulteriori informazioni suJSON, consulta json.org.



JSONrappresenta 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 JSON oggetto che contiene una serie 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 l'altra che specifica una 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"}
];
```

JSONcome parametri dell'oggetto di servizio

Ecco un esempio di JSON utilizzo semplice per definire i parametri di una chiamata a un oggetto AWS Lambda di 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 di un oggetto di servizio, fornite l'JSONoggetto 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 };
};
```

SDKper esempi di JavaScript codice

Gli argomenti di questa sezione contengono esempi di come utilizzare AWS SDK for JavaScript i APIs 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, anziché semplici frammenti di codice che coprano solo singole chiamate. API Per istruzioni, consulta la sezione Codice di creazione nelle linee guida per i contributi su. GitHub

> **Important**
>
> Questi esempi utilizzano la sintassi di ECMAScript6 importazione/esportazione.

- 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

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 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"`dal`package.json`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

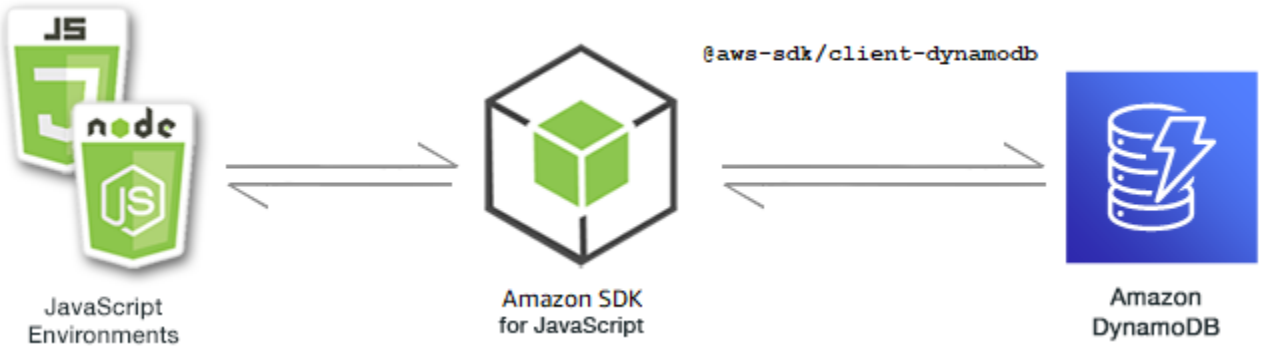
```
// Create service client module using CommonJS syntax.
 const { S3Client } = require("@aws-sdk/client-s3");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
 // Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
 module.exports ={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 SQL No cloud 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.



Il JavaScript API for DynamoDB viene esposto attraverso `DynamoDB` le classi`DynamoDBStreams`, `DynamoDB.DocumentClient` e client. Per ulteriori informazioni sull'utilizzo delle classi client DynamoDB, vedere l'utilità Class: DynamoDB, Class: D e Class: DynamoDB nella ynamoDBStreams Guida di riferimento. API

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 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 il comando SDK for JavaScript per creare e lavorare con le tabelle utilizzando questi metodi della classe `DynamoDB` 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.
- Installazione 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 and Tools Reference Guide.AWS SDKs

> **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 configurarli SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Create un JSON oggetto contenente 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 da fornire. 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 è disponibile qui su GitHub.

Elencare i tavoli

Crea un modulo Node.js con il nome del file `list-tables.js`. Assicurati di configurarlo 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 JSON oggetto contenente i parametri necessari per elencare le tue 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 è disponibile qui su GitHub.

Descrizione di una tabella

Crea un modulo Node.js con il nome del file describe-table.js. Assicurati di configurarlo 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 JSON oggetto 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 è disponibile qui su GitHub.

Eliminazione di una tabella

Crea un modulo Node.js con il nome del file `delete-table.js`. Assicurati di configurarlo 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 JSON oggetto contenente i parametri necessari per eliminare una tabella, che in questo esempio include il nome della tabella fornita 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 è disponibile 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 AWS SDKs and Tools Reference Guide.
- 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 configurarli SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Create un JSON oggetto contenente i parametri necessari per aggiungere un elemento, 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 è disponibile qui su GitHub.

Aggiornamento di un elemento

Crea un modulo Node.js con il nome del file `update-item.js`. Assicurati di configurarlo 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 JSON oggetto 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 è disponibile qui su GitHub.

Ottenimento di un elemento

Crea un modulo Node.js con il nome del file `get-item.js`. Assicurati di configurarlo 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 JSON oggetto contenente i parametri necessari per ottenere un elemento, che in questo esempio include il nome della tabella, il nome e il valore della chiave dell'elemento che stai ricevendo e un'espressione di proiezione che identifica l'attributo dell'elemento che desideri 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 è disponibile qui su GitHub.

Eliminazione di un elemento

Crea un modulo Node.js con il nome del file `delete-item.js`. Assicurati di configurarlo 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 JSON oggetto contenente i parametri necessari per eliminare un elemento, che in questo esempio include il nome della tabella e sia il nome della chiave che il valore dell'elemento che stai 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 è disponibile 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 il comando SDK for JavaScript per 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 AWS SDKs and Tools Reference Guide.
- 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 configurarlo SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Create un JSON oggetto contenente i parametri necessari per ottenere un batch di elementi, che in questo esempio include il nome di una o più tabelle da cui leggere, i valori delle chiavi da leggere in ogni 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 è disponibile qui su GitHub.

Scrittura di articoli in batch

Crea un modulo Node.js con il nome del file `batch-write-item.js`. Assicurati di configurarlo SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere

a DynamoDB, crea DynamoDB un oggetto servizio client. Create un JSON oggetto contenente i parametri necessari per ottenere un batch di elementi, che in questo esempio include la tabella in cui desiderate scrivere gli elementi, le chiavi da 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 è disponibile 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 il comando SDK for JavaScript per 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 AWS SDKs and Tools Reference Guide.
- 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 configurarli SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Create un JSON oggetto contenente i parametri necessari per interrogare la tabella, che in questo esempio include il nome della tabella, quello `ExpressionAttributeValues` necessario alla query, `KeyConditionExpression` che utilizza tali valori per definire gli elementi restituiti dalla query e i nomi dei valori degli attributi da restituire per ogni elemento. 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 è disponibile qui su GitHub.

Scansione di una tabella

Crea un modulo Node.js con il nome del file `scan.js`. Assicurati di configurarlo SDK come mostrato in precedenza, incluso il download dei client e dei pacchetti richiesti. Per accedere a DynamoDB, crea `DynamoDB` un oggetto servizio client. Create un JSON oggetto contenente i parametri necessari per la scansione degli elementi nella tabella, che in questo esempio include il nome della tabella, l'elenco dei valori degli attributi da restituire per ogni elemento corrispondente e un'espressione per filtrare il set di risultati per trovare gli elementi contenenti una frase specificata. 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 on. README GitHub Per ulteriori informazioni sulla programmazione con Amazon DynamoDB, consulta Programming Amazon DynamoDB JavaScript with 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 il comando SDK for per JavaScript interrogare e 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 AWS SDKs and Tools Reference Guide.
- Crea una tabella DynamoDB a cui puoi accedere agli elementi. Per ulteriori informazioni sulla creazione di una tabella DynamoDB utilizzando SDK il JavaScript comando for, 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 configurarli 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 JSON oggetto 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, inserisci quanto segue al prompt dei comandi.

```
node get.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Inserimento di una voce in una tabella

Crea un modulo Node.js con il nome del file `put.js`. Assicurati di configurarlo 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. Create un JSON oggetto 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 l'hashkey e il valore e i nomi e i valori degli 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, inserisci quanto segue al prompt dei comandi.

```
node put.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Aggiornamento di una voce in una tabella

Crea un modulo Node.js con il nome del file `update.js`. Assicurati di configurarlo 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 JSON oggetto 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 può essere trovato qui su GitHub.

Esecuzione di query su una tabella

Crea un modulo Node.js con il nome del file `query.js`. Assicurati di configurarlo 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`. Create un JSON oggetto contenente i parametri necessari per interrogare la tabella, che in questo esempio include il nome della tabella, i parametri `ExpressionAttributeValues` necessari alla query e un oggetto `KeyConditionExpression` che utilizzi tali valori per definire gli elementi restituiti dalla query. 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, inserisci quanto segue al prompt dei comandi.

```
node query.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione di una voce da una tabella.

Crea un modulo Node.js con il nome del file `delete.js`. Assicurati di configurarlo 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` Create un JSON oggetto 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 della chiave hash dell'elemento che desiderate 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, inserisci quanto segue al prompt dei comandi.

```
node delete.js
```

Questo codice di esempio può essere trovato 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 AWS SDK 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 con il ruolo IAM associato. 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 e fornire implementazioni di AWS infrastrutture in modo prevedibile e ripetuto. Per ulteriori informazioni in merito AWS CloudFormation, consulta la Guida per l'utente.AWS CloudFormation

Per creare lo AWS CloudFormation stack:

1. Installa e configura le AWS CLI seguenti istruzioni contenute nella Guida per l'AWS CLI utente.
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 AWS Cloud Development Kit (AWS CDK) gli sviluppatori.

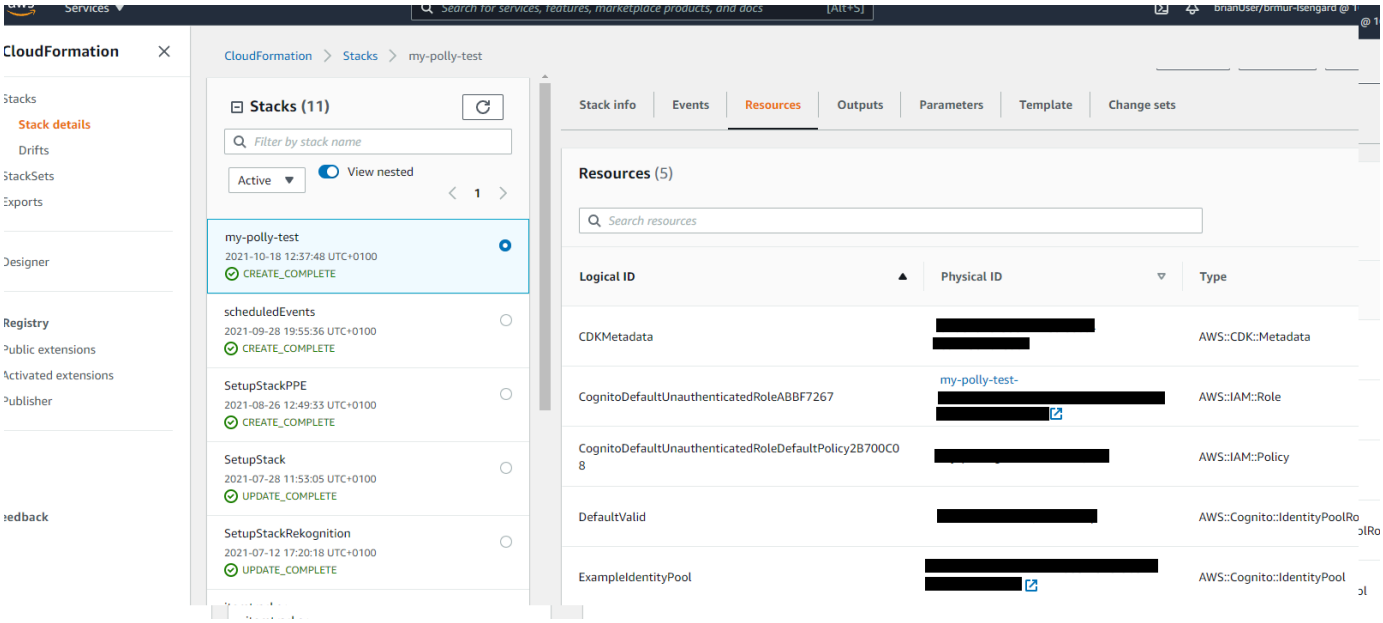
3. Esegui il comando seguente dalla riga di comando, sostituendo *STACK_NAME con un nome* univoco per lo stack.

> **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 CloudFormation utente.

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 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.

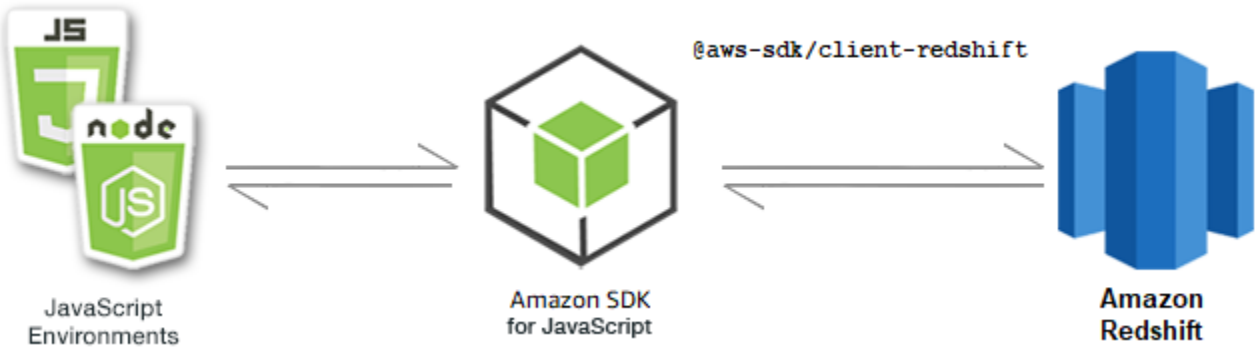
Chiama il `StartSpeechSynthesisCommand` metodo dell'oggetto del servizio client Amazon Polly, sintetizza il messaggio vocale e caricalo nel bucket Amazon S3.

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

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

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

Questo codice di esempio è disponibile 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();
```

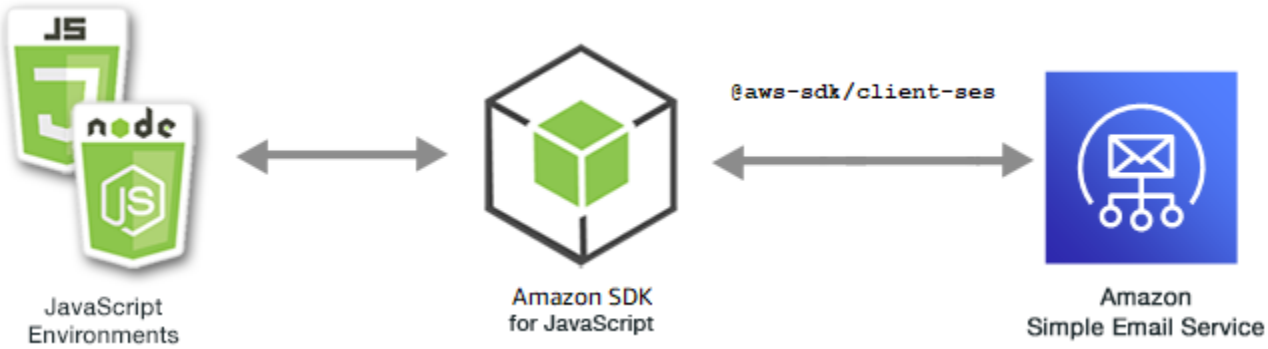
Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui. GitHub

Esempi di Amazon Simple Email Service

Amazon Simple Email Service (AmazonSES) è un servizio di invio e-mail basato sul cloud progettato per aiutare gli esperti di 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.



Il JavaScript API for Amazon SES viene esposto tramite la classe SES client. Per ulteriori informazioni sull'utilizzo della classe SES client Amazon, consulta Class: SES nel API riferimento.

Argomenti

- Gestione delle SES identità Amazon
- Lavorare con i modelli di e-mail in Amazon SES
- Invio di e-mail tramite Amazon SES

Gestione delle SES identità Amazon

Questo esempio di codice di Node.js illustra:

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

Un'SESidentità Amazon è un indirizzo e-mail o un dominio che Amazon SES utilizza per inviare e-mail. Amazon ti SES richiede di verificare 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 AmazonSES, consulta la sezione Verifica degli indirizzi e-mail e dei domini in Amazon nella Amazon SES Simple Email Service Developer Guide. Per informazioni sull'autorizzazione all'invio in AmazonSES, consulta Panoramica dell'autorizzazione SES all'invio di Amazon.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per verificare e gestire SES le identità Amazon. I moduli Node.js utilizzano il form SDK JavaScript per verificare gli indirizzi e-mail e i domini, utilizzando questi metodi della classe `SES` 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 strumenti AWS SDKs e agli strumenti.

> **Important**
>
> Questi esempi mostrano 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 AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_listidentities.js`. Configura 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 SES servizio Amazon, passando l'oggetto parameters.

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

> **Note**
>
> Replace (Sostituisci) *`IdentityType`* con il tipo di identità, che può essere "EmailAddress" o «Dominio».

```
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 AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_verifyemailidentity.js`. Configura 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 SES del servizio client Amazon, passando i parametri.

> **Note**
>
> Replace (Sostituisci) *`EMAIL_ADDRESS`* 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 AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_verifydomainidentity.js`. Configura 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 SES del servizio client Amazon, 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**
>
> Replace (Sostituisci) *DOMAIN_NAME* con il nome di dominio.

```
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, immettere 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 AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_deleteidentity.js`. Configura 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 `request` per richiamare un oggetto SES del servizio client Amazon, 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**
>
> Replace (Sostituisci) *IDENTITY_EMAIL* con l'e-mail 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.

Lavorare con i 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 ti 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 AmazonSES, consulta Invio di e-mail personalizzate utilizzando Amazon SES API 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 SDK for 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 strumenti AWS SDKs e agli strumenti.

> **Important**
>
> Questi esempi mostrano 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 AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_listtemplates.js`. Configura 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 SES del servizio client Amazon, 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.

```
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, usa un modulo Node.js per ottenere un modello di email da usare con AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_gettemplate.js`. Configura 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 SES del servizio client Amazon, 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**
>
> Replace (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 (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Per eseguire l'esempio, immettere 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 AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_createtemplate.js`. Configura 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 SES del servizio client Amazon, 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**
>
> Replace (Sostituisci) *`TEMPLATE_NAME`* con un nome per il nuovo modello, *`HtmlPart`* con il contenuto HTML taggato dell'e-mail e *`SubjectPart`* con l'oggetto 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, immettere quanto segue al prompt dei comandi. Il modello viene aggiunto ad AmazonSES.

```
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 AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_updatetemplate.js`. Configura 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 SES servizio Amazon, 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**
>
> Replace (Sostituisci) *`TEMPLATE_NAME`* con un nome del modello e *`HTML_PART`* con il contenuto HTML taggato 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, immettere 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 AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_deletetemplate.js`. Configura 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 SES servizio Amazon, 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**
>
> Replace (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, immettere 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:

- Invia un SMS o HTML un'e-mail.
- 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.

Amazon SES API 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 maggiori dettagli, consulta Invio di e-mail formattate tramite Amazon SES API e Invio di e-mail non elaborate tramite Amazon SES API.

Lo scenario

In questo esempio, utilizzi una serie di moduli Node.js per inviare e-mail in diversi modi. I moduli Node.js utilizzano SDK for 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 strumenti AWS SDKs e agli strumenti.

> **Important**
>
> Questi esempi mostrano 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 è ancora nella SES sandbox di Amazon, puoi inviare messaggi solo a indirizzi o domini verificati o a indirizzi e-mail associati ad Amazon SES Mailbox Simulator. 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:, un indirizzo CC: o un indirizzoBCC:. 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: eBCC:. 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, usa un modulo Node.js per inviare e-mail con AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_sendemail.js`. Configura 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 testo e HTML formati semplici, al `SendEmailCommand` metodo della classe `SES` client. Per chiamare il `SendEmailCommand` metodo, richiama un oggetto di SES servizio Amazon, 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**
>
> Replace (Sostituisci) *`toAddress`* con l'indirizzo a cui inviare l'e-mail e *`fromAddress`* 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 (caught) {
```

```
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi. L'e-mail è 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, usa un modulo Node.js per inviare e-mail con AmazonSES. Crea un modulo Node.js con il nome del file `ses_sendtemplatedemail.js`. Configura 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, il corpo dell'e-mail in testo semplice e i HTML formati, al `SendTemplatedEmailCommand` metodo della classe `SES` client. Per chiamare il `SendTemplatedEmailCommand` metodo, richiama un oggetto SES del servizio client Amazon, 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**
>
> Replace (Sostituisci) *REGION* con la tua AWS regione, *USER* con il nome e l'indirizzo e-mail a cui inviare l'e-mail, *VERIFIED_EMAIL* 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 (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi. L'e-mail è 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, usa un modulo Node.js per inviare e-mail con AmazonSES.

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 SES client Amazon. Replace (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 su GitHub.

Crea un modulo Node.js con il nome del file `ses_sendbulktemplatedemail.js`. Configura 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 testo e HTML formati semplici, al `SendBulkTemplatedEmailCommand` metodo della classe `SES` client. Per chiamare il `SendBulkTemplatedEmailCommand` metodo, richiama un oggetto di SES servizio Amazon, 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**
>
> Replace (Sostituisci) *`USERS`* con i nomi e gli indirizzi e-mail a cui inviare l'e-mail, *`VERIFIED_EMAIL_1`* con l'indirizzo e-mail da cui inviare l'e-mail, e *`TEMPLATE_NAME`* con il nome del modello.

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

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

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

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

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map each
 user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
```

```
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Per eseguire l'esempio, immettere quanto segue al prompt dei comandi. L'e-mail è 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.

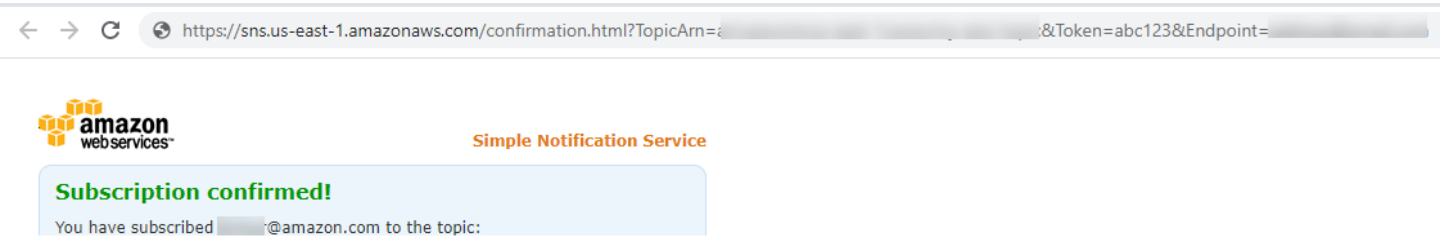
```
import { SNSClient } from "@aws-sdk/client-sns";

// 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 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

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.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
```

```
export { transcribeClient };
```

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.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

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 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

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, consulta start MedicalTranscription Job.

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-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 JavaScriptPer ulteriori informazioni, consulta Command. ListTranscription MedicalJobs

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 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'EC2istanza Amazon

Uno scenario comune per l'utilizzo di Node.js con il SDK JavaScript modulo consiste nel configurare ed eseguire un'applicazione Web Node.js su un'istanza Amazon Elastic Compute Cloud (AmazonEC2). In questo tutorial, creerai un'istanza Linux, ti connetterai ad essa utilizzando SSH e quindi installerai Node.js per eseguirla su quell'istanza.

Prerequisiti

Questo tutorial presuppone che tu abbia già avviato un'istanza Linux con un DNS nome pubblico raggiungibile da Internet e alla quale puoi connetterti utilizzando. SSH Per ulteriori informazioni, consulta la Fase 1: Avvio di un'istanza nella Amazon EC2 User Guide.

> **Important**
>
> Usa Amazon Linux 2023 Amazon Machine Image (AMI) quando avvii una nuova EC2 istanza Amazon.

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.

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. Connect alla propria istanza Linux `ec2-user` utilizzandoSSH.
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 LTS versione 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 EC2 sessione Amazon corrente. Se riavvii la CLI sessione, devi usare 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'EC2istanza Amazon una volta ottenuta la configurazione che desideri mantenere, come descritto nel seguente argomento.

Creazione di un'immagine di una macchina Amazon (AMI)

Dopo aver installato Node.js su un'EC2istanza Amazon, puoi creare un'Amazon Machine Image (AMI) da quell'istanza. La creazione di un file AMI semplifica il provisioning di più EC2 istanze Amazon con la stessa installazione di Node.js. Per ulteriori informazioni sulla creazione di un AMI da un'istanza esistente, consulta Creating an amazon EBS -backed Linux AMI nella Amazon EC2 User Guide.

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.

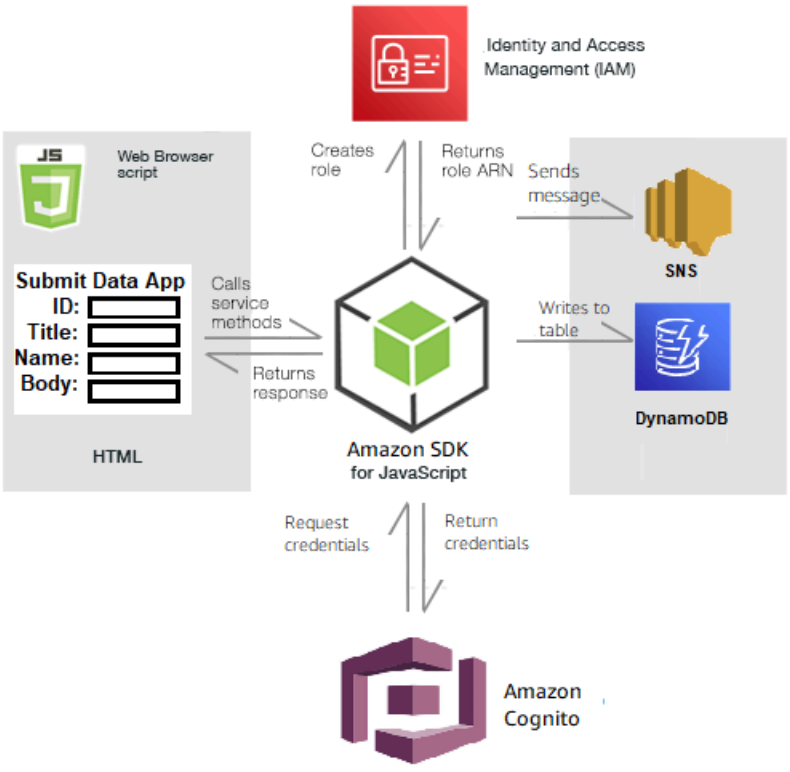
Crea un'app per inviare dati a DynamoDB

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.

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.

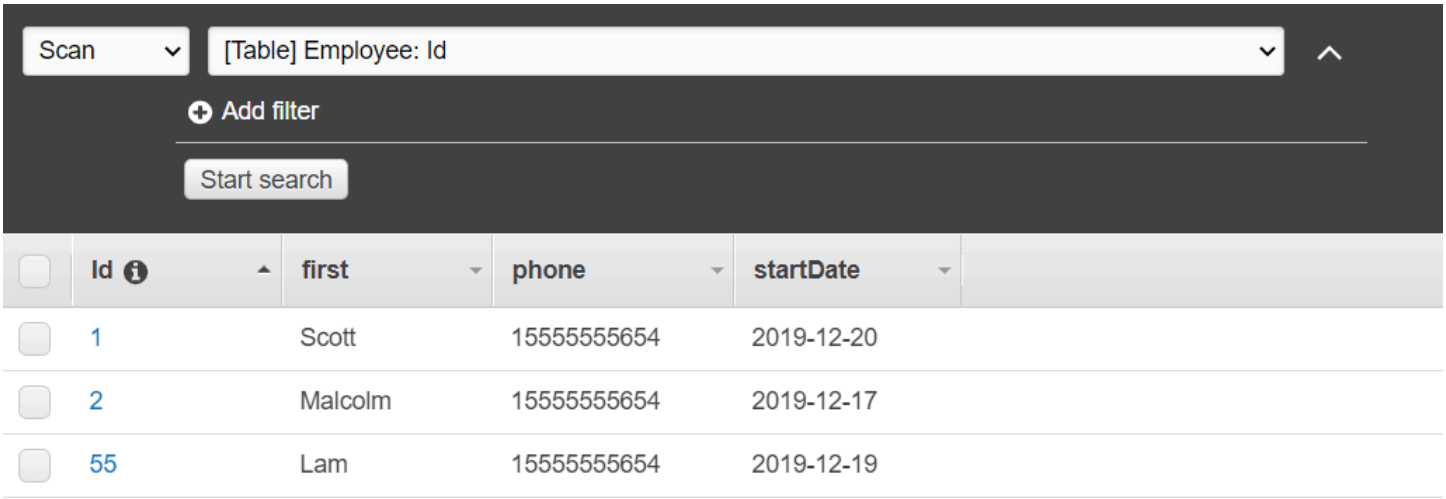
Il completamento dell'esempio dovrebbe richiedere circa 20 minuti.

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

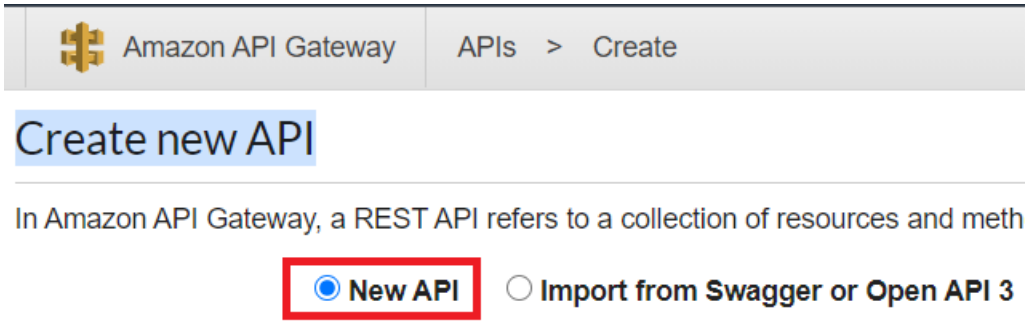
Per creare l'app:

1. Crea la restante API
2. Prova il metodo API Gateway
3. Implementa il metodo API Gateway

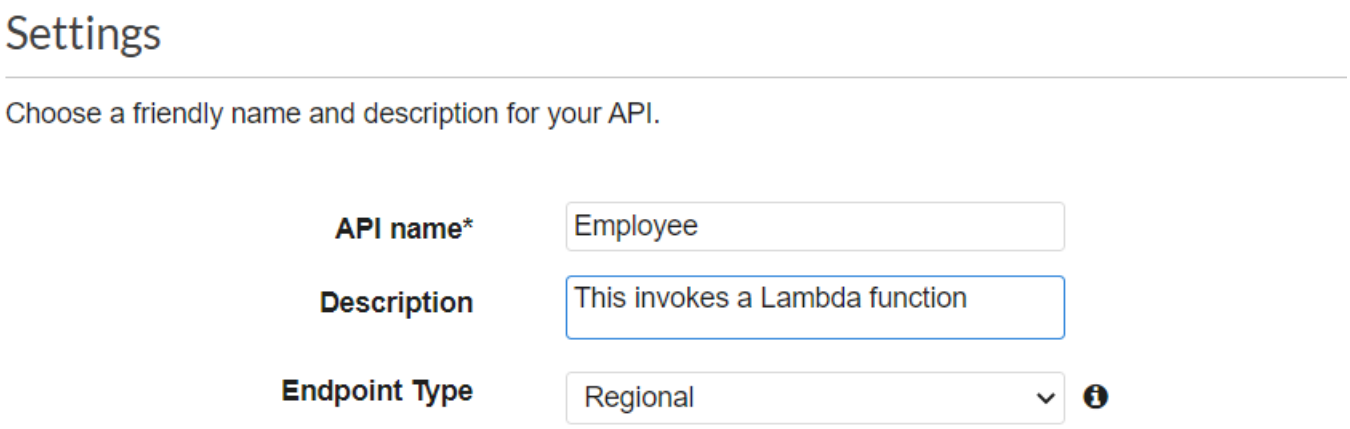
Crea la restante API

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.

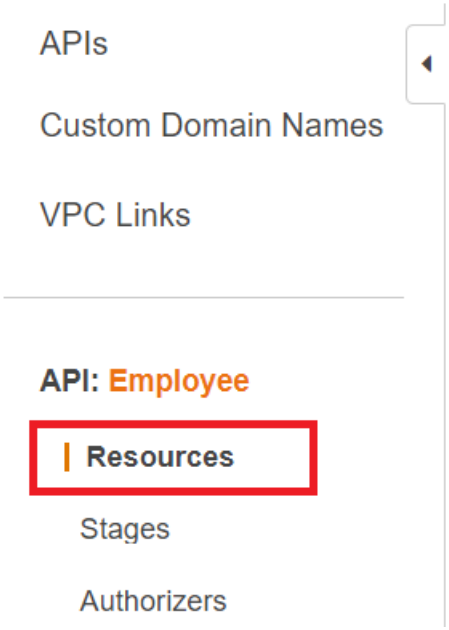
1. Accedi alla console Amazon API Gateway.
2. In Rest API, scegli Build.
3. Seleziona Nuova API.



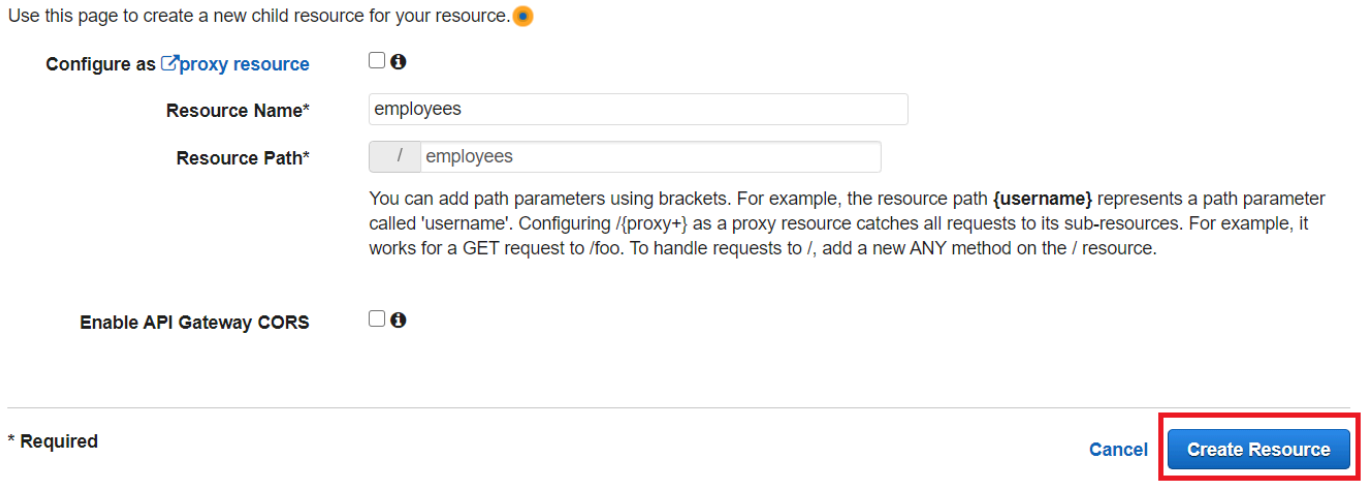
4. Specificate Employee come nome dell'API e fornite una descrizione.



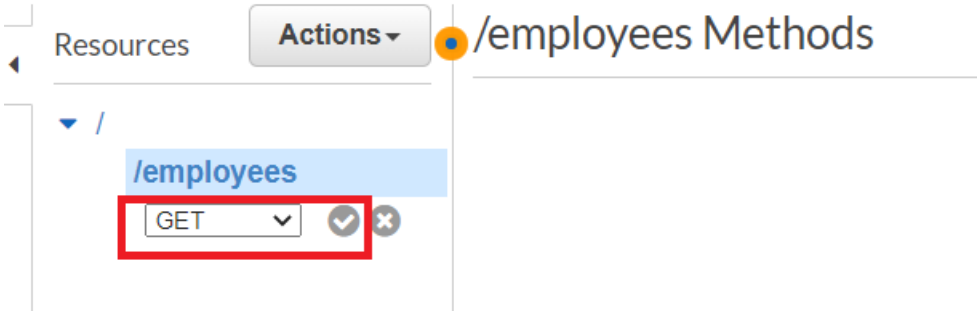
5. Seleziona Create API (Crea API).
6. Scegli Risorse nella sezione Dipendenti.



7. Nel campo del nome, specifica i dipendenti.
8. Selezionare Create Resources (Crea risorse).
9. Dal menu a discesa Azioni, scegli Crea risorse.



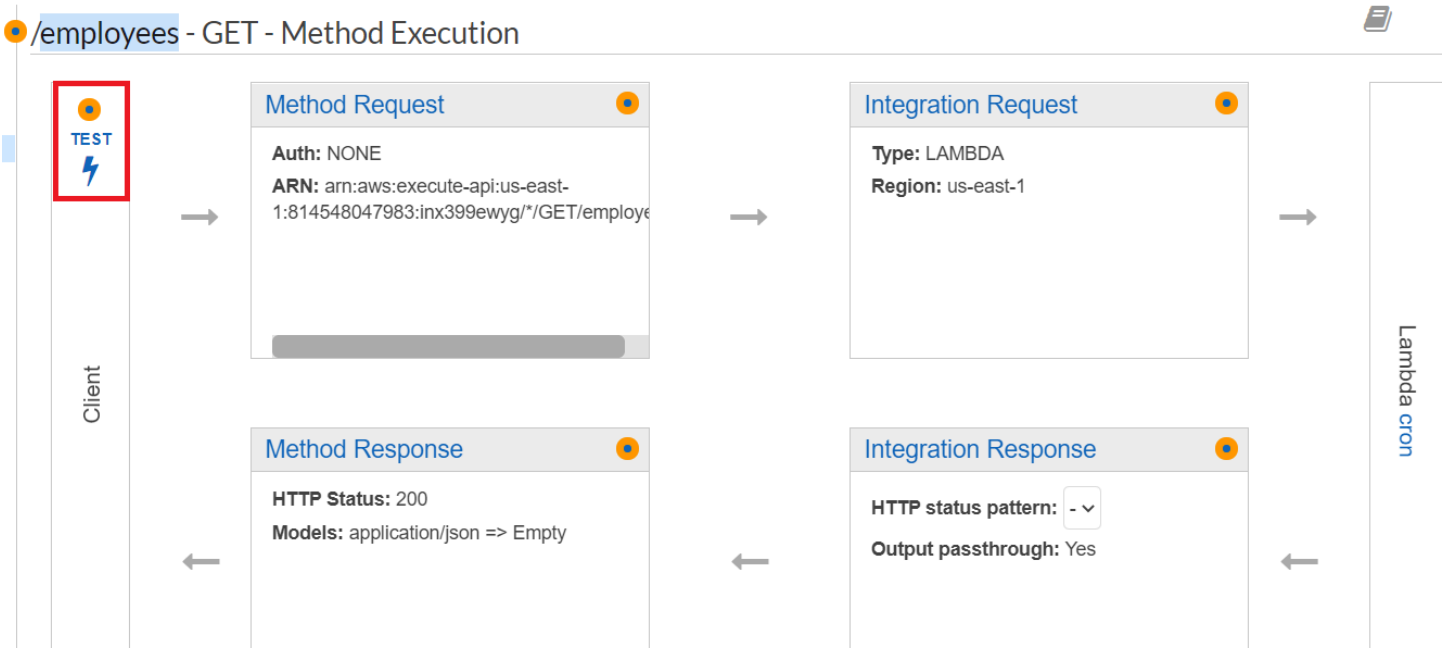
10. Scegli /employees, seleziona Crea metodo dal menu Azioni, quindi seleziona GET dal menu a discesa sotto /employees. Selezionare l'icona del segno di spunta.



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.

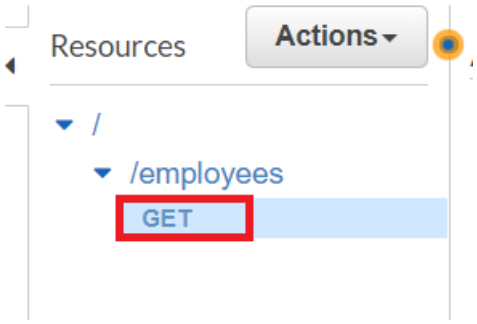


Una volta richiamata la funzione Lambda, è possibile visualizzare il file di registro per visualizzare un messaggio di successo.

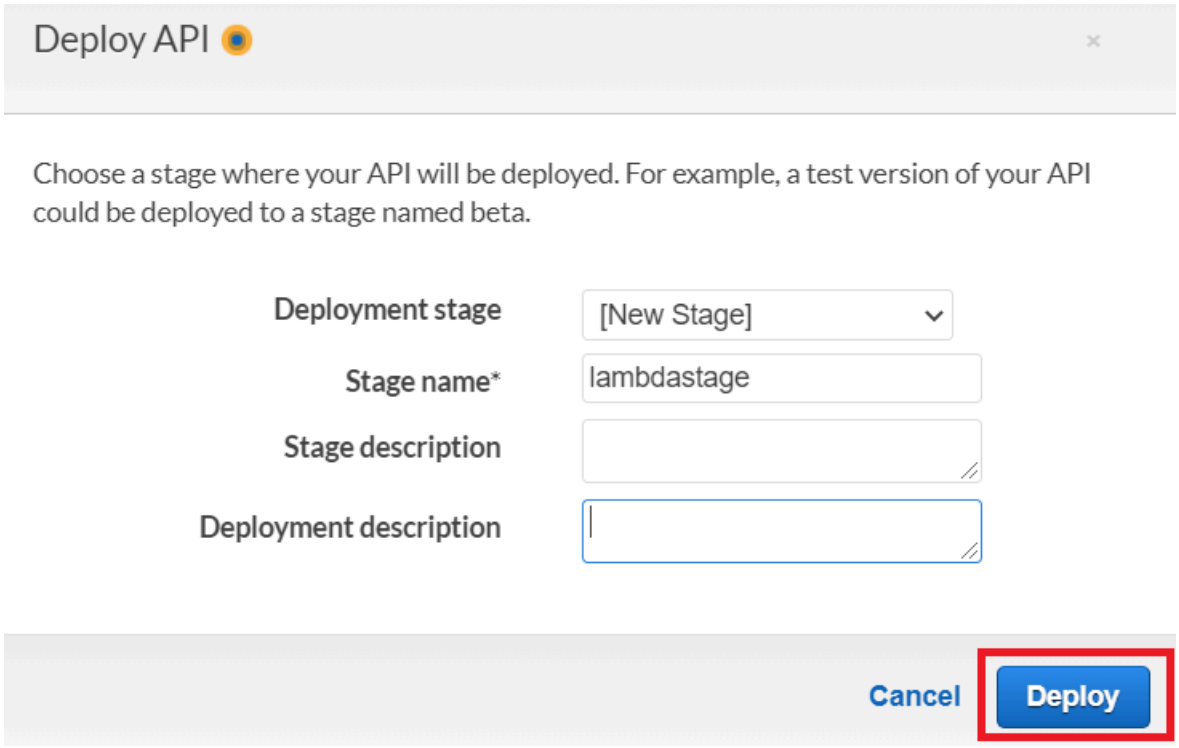
Implementa il metodo API Gateway

Una volta completato il test, puoi implementare il metodo dalla console di Amazon API Gateway.

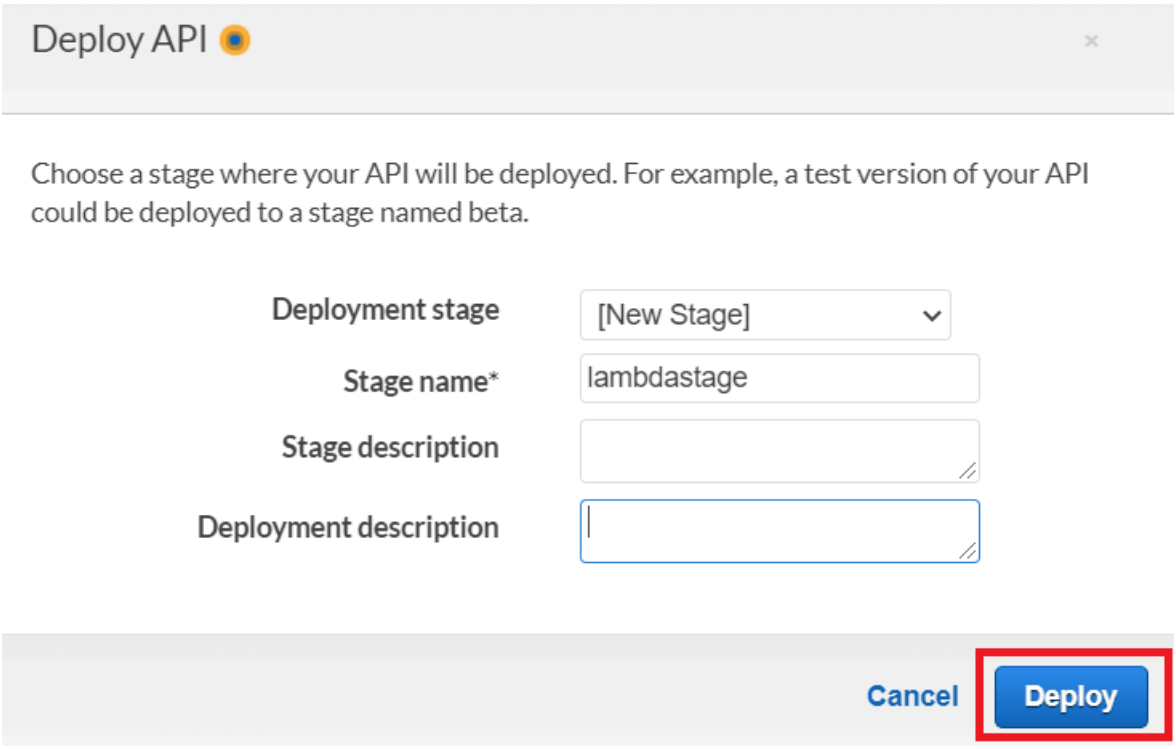
1. Scegli Get.



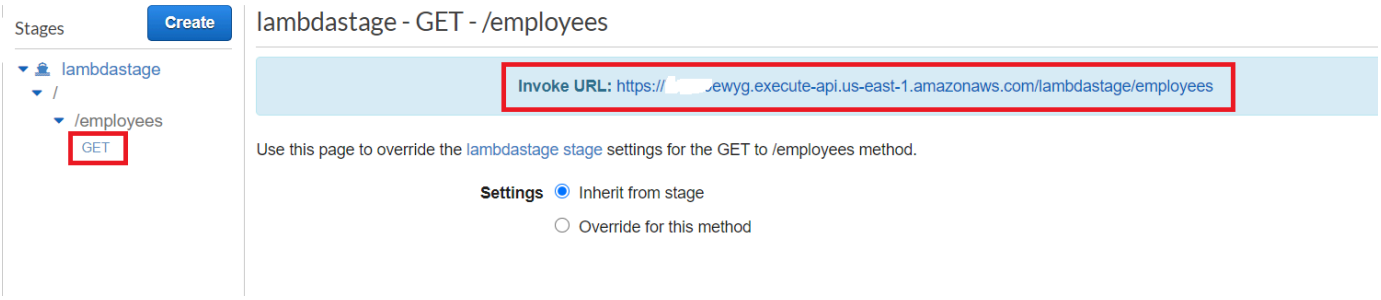
2. Dal menu a discesa Azioni, seleziona Deploy API.



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 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.

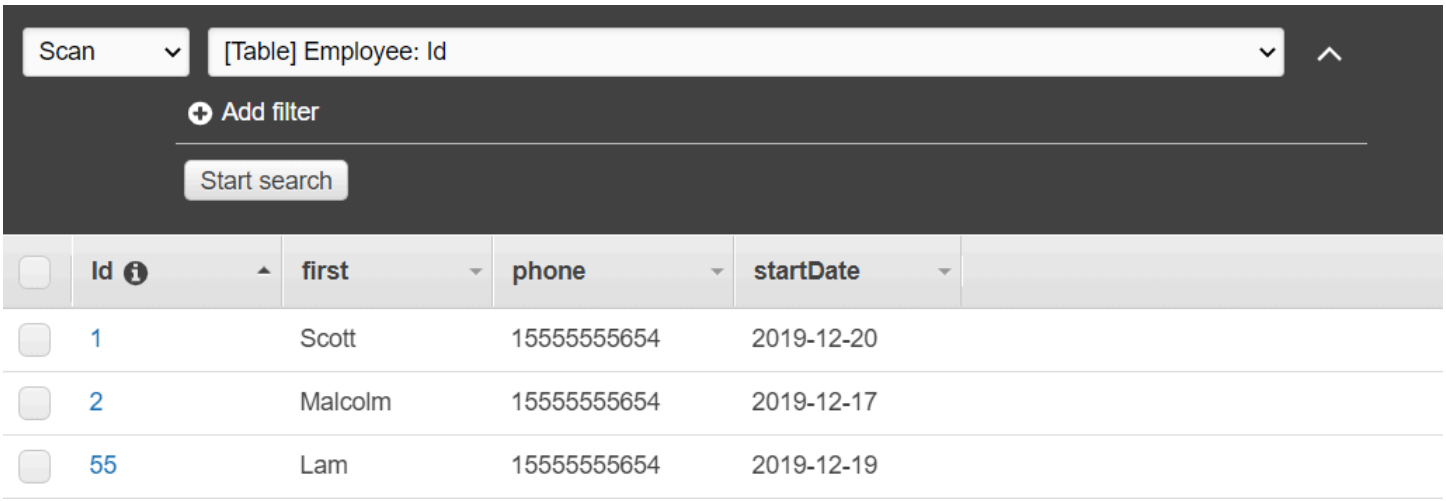
Il completamento di questo tutorial richiede circa 20 minuti.

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.

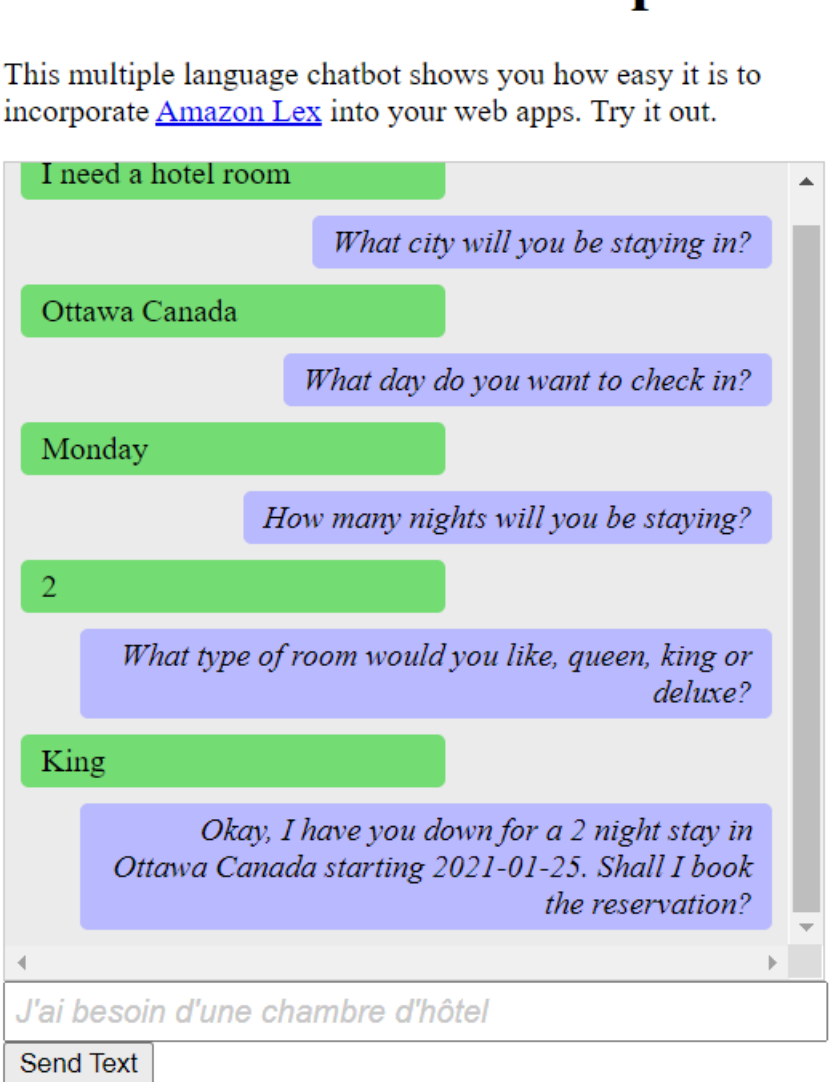
Eliminare le risorse

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:

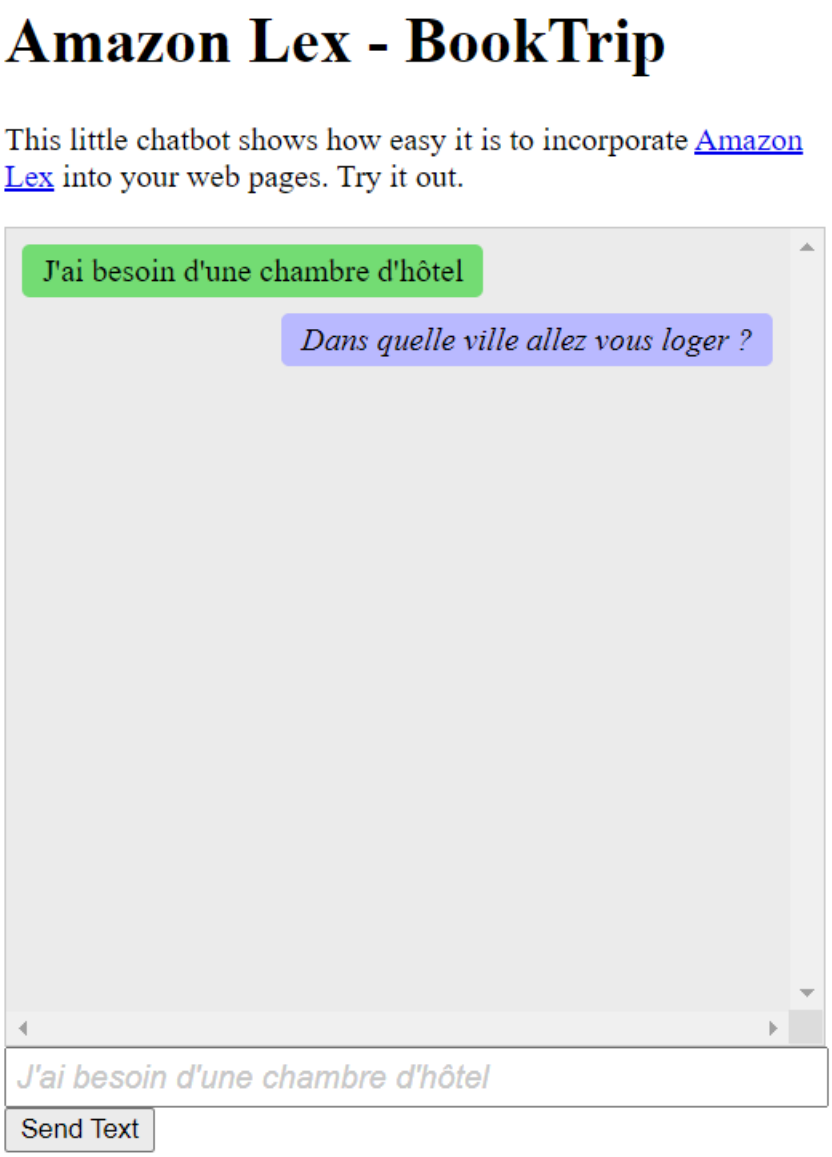
1. Apri la AWS CloudFormation console.
2. Nella pagina Stacks, seleziona lo stack.
3. Scegli Delete (Elimina).

Creazione di un chatbot Amazon Lex

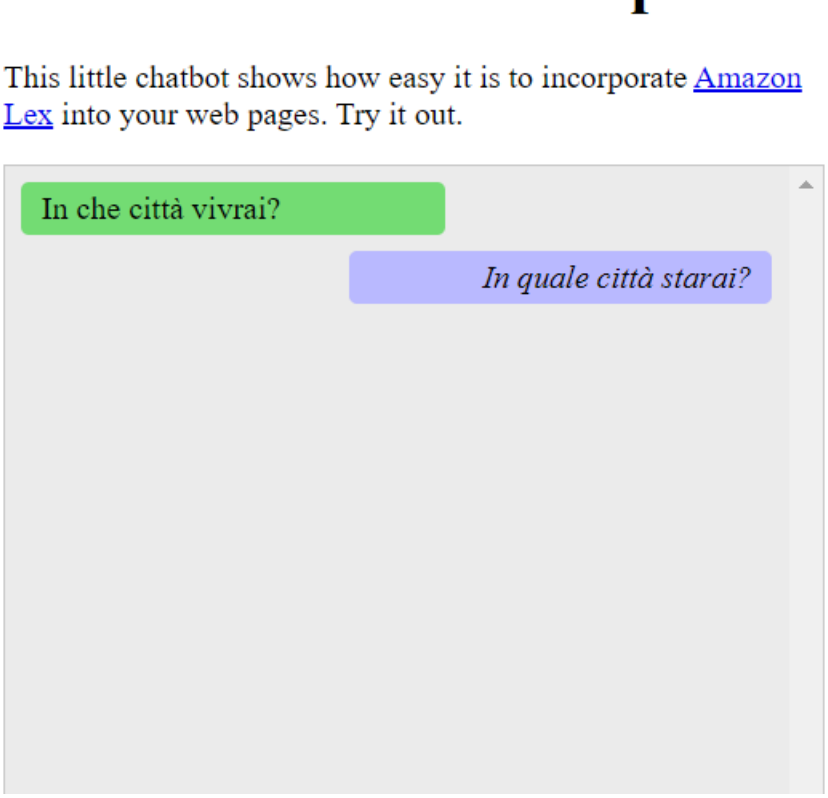
Puoi creare un chatbot Amazon Lex all'interno di un'applicazione Web per coinvolgere i visitatori del tuo sito Web. Un chatbot di Amazon Lex è una funzionalità che esegue conversazioni in chat online con gli utenti senza fornire un contatto diretto con una persona. Ad esempio, l'illustrazione seguente 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.

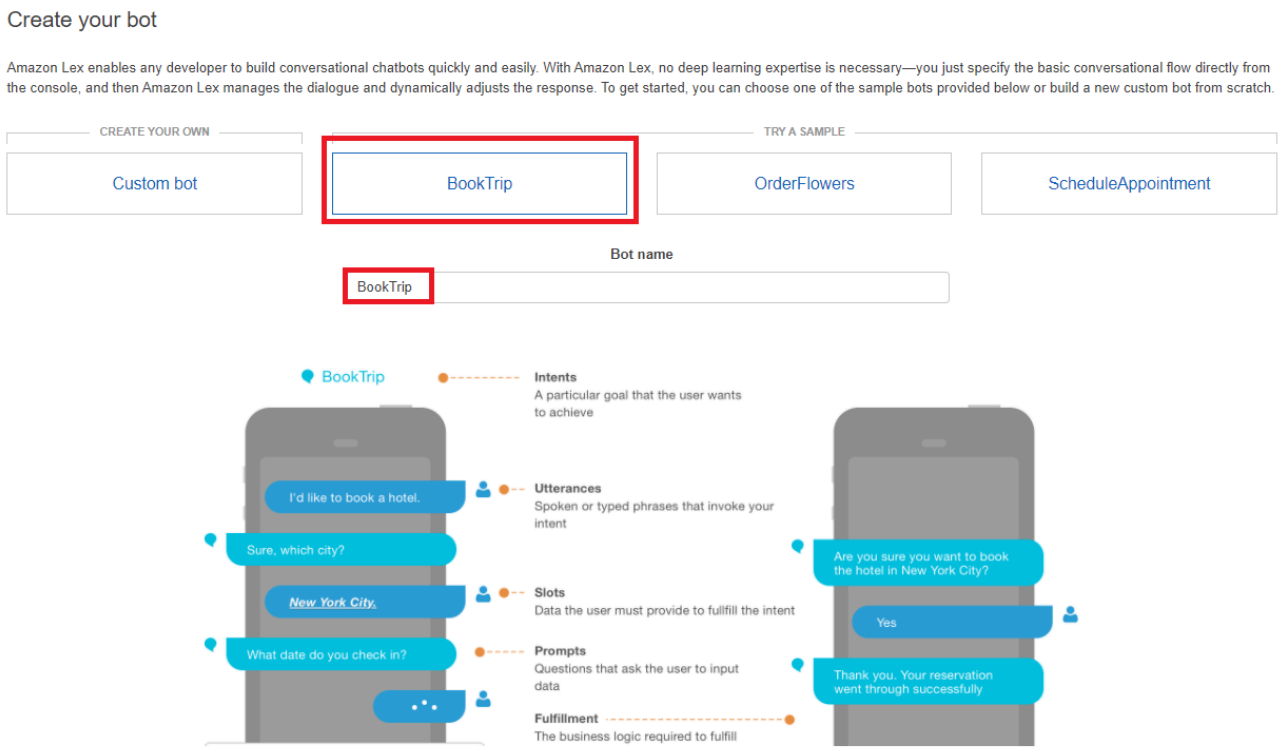
   Per visualizzare le risorse create, apri la console Amazon Lex, scegli lo stack e seleziona la scheda Risorse.

Creazione di un bot Amazon Lex

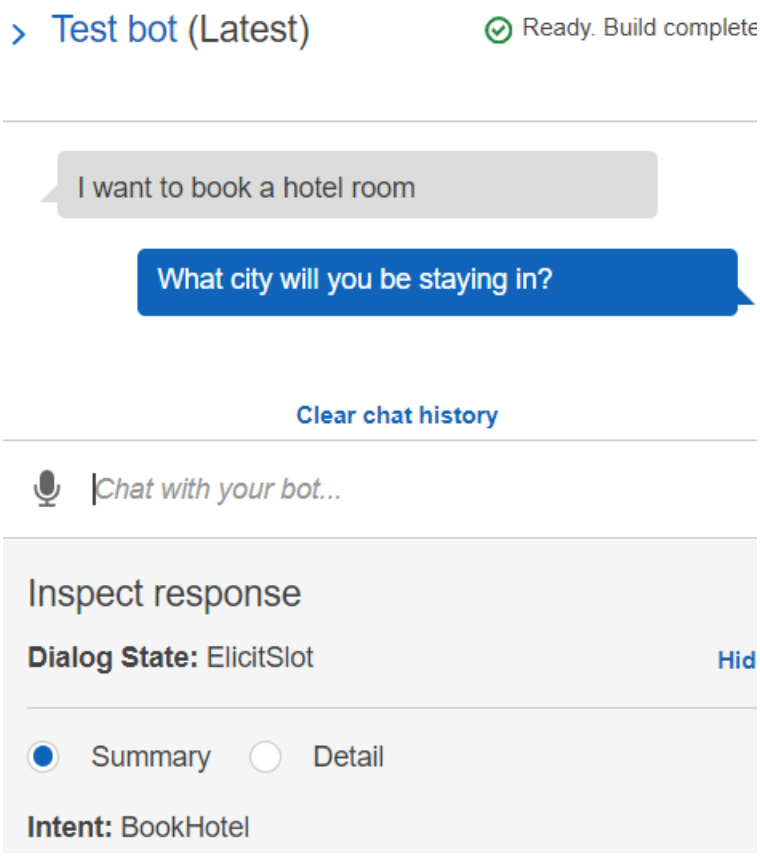
> **Important**
>
> Usa la versione 1 della console Amazon Lex per creare il bot. Questo esempio non funziona con i bot creati utilizzando V2.

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.

SDKper JavaScript esempi di codice (v3)

Gli esempi di codice in questo argomento mostrano come utilizzare AWS SDK for JavaScript (v3) con. AWS

Gli elementi di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto nei relativi scenari.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Servizi

- APIEsempi di gateway che utilizzano SDK for JavaScript (v3)
- Esempi di Aurora che utilizzano SDK for JavaScript (v3)
- 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 Amazon Comprehend con SDK for JavaScript (v3)
- Esempi di Amazon DocumentDB con SDK for JavaScript (v3)
- Esempi di DynamoDB SDK con JavaScript for (v3)
- EC2Esempi di Amazon che utilizzano SDK for JavaScript (v3)
- Elastic Load Balancing - Esempi della versione 2 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)
- IAMesempi che usano SDK for JavaScript (v3)
- Esempi di Kinesis con SDK for JavaScript (v3)
- Esempi di Lambda che utilizzano SDK for JavaScript (v3)
- Esempi di Amazon Lex con SDK for JavaScript (v3)
- MSKEsempi di Amazon che utilizzano SDK for JavaScript (v3)
- Esempi di Amazon Personalize usando SDK for JavaScript (v3)
- Esempi di Amazon Personalize Events con 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 Polly con SDK for JavaScript (v3)
- RDSEsempi di Amazon che utilizzano SDK for JavaScript (v3)
- Esempi di Amazon RDS Data Service con SDK for JavaScript (v3)
- Esempi di Amazon Redshift con SDK for JavaScript (v3)
- Esempi di Amazon Rekognition SDK con for (v3) JavaScript
- Esempi di Amazon S3 con SDK for JavaScript (v3)
- Esempi di S3 Glacier che utilizzano for (v3) SDK JavaScript
- SageMaker esempi che utilizzano SDK for JavaScript (v3)
- Esempi di Secrets Manager che utilizzano SDK for JavaScript (v3)
- SESEsempi di Amazon che utilizzano SDK for JavaScript (v3)
- SNSEsempi di Amazon che utilizzano SDK for JavaScript (v3)
- SQSEsempi di Amazon che utilizzano 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 Textract con SDK for JavaScript (v3)
- Esempi di Amazon Transcribe SDK con JavaScript for (v3)
- Esempi di Amazon Translate con SDK for JavaScript (v3)

APIEsempi di gateway 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 API Gateway.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDKper 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, vedi l'esempio completo su GitHub.
>
> Per approfondire l'origine di questo esempio, consulta il post su AWS Community.
>
> Servizi utilizzati in questo esempio
>
> - APIGateway
> - DynamoDB
> - Lambda
> - Amazon Rekognition
> - Amazon S3

- Amazon SNS

Usa API Gateway per richiamare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da Amazon API Gateway.

SDKper JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando il runtime Lambda JavaScript . API Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare una funzione Lambda richiamata da API Amazon Gateway che scansiona una tabella Amazon DynamoDB per gli anniversari di lavoro e utilizza Amazon Simple SNS Notification Service (Amazon) per inviare un messaggio di testo ai dipendenti in cui si congratula per la data del primo anniversario.

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

- APIGateway
- DynamoDB
- Lambda
- Amazon SNS

Esempi di Aurora 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 Aurora.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

Il seguente esempio di codice mostra come creare un'applicazione Web che tiene traccia degli elementi di lavoro in un database Amazon Aurora Serverless e utilizza Amazon Simple Email Service (AmazonSES) per inviare report.

SDKper 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 (AmazonSES). 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. AWS servizi
- Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
- Invia un report via e-mail sugli elementi di lavoro filtrati utilizzando AmazonSES.
- Distribuisci e gestisci risorse di esempio con lo script incluso. AWS CloudFormation

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
- Servizio RDS dati Amazon
- Amazon SES

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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`AttachLoadBalancerTargetGroups`

Il seguente esempio di codice mostra come utilizzare`AttachLoadBalancerTargetGroups`.

SDKper JavaScript (v3)

> **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.

```
const client = new AutoScalingClient({});
await client.send(
  new AttachLoadBalancerTargetGroupsCommand({
    AutoScalingGroupName: NAMES.autoScalingGroupName,
    TargetGroupARNs: [state.targetGroupArn],
  }),
);
```

- Per API i dettagli, vedi AttachLoadBalancerTargetGroups AWS SDK for JavaScriptAPIReference.

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 Amazon EC2 Auto Scaling per creare istanze Amazon Elastic Compute Cloud (AmazonEC2) basate su un modello di avvio e per mantenere il numero di istanze in un intervallo specificato.
- Gestisci e distribuisci HTTP le richieste 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 EC2 istanza 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 di integrità aggiornando AWS Systems Manager i parametri.

SDKper JavaScript (v3)

> **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.

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.

```
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 { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 () => {
  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,
      },
    }),
  );
}),
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) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    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,
      }),
    );
    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) => {
    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] },
    );
  }),
```

```
  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) => {
    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 },
        ],
      }),
    );
    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) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
```

```
      }),
    );
  }),
  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,
      );
    }
  }),
  saveState,
];
```

Crea i passaggi per eseguire la demo.

```
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) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
```

```
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
```

```
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
```

```
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
```

```
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );

      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",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Crea i passaggi per distruggere tutte le risorse.

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} 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 { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 = [
  loadState,
  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 {
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } 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 {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } 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(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    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;
    }
  }),
  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,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
```

```
];

/**
 * @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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - 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. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile 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.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListFoundationModels AWS SDK for JavaScriptAPIReference.

Argomenti

- Azioni

Azioni

`GetFoundationModel`

Il seguente esempio di codice mostra come utilizzare`GetFoundationModel`.

SDKper JavaScript (v3)

> **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.

Ottieni dettagli su un modello di base.

```
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 API i dettagli, vedere GetFoundationModelin AWS SDK for JavaScript APIReference.

`ListFoundationModels`

Il seguente esempio di codice mostra come utilizzare`ListFoundationModels`.

SDKper JavaScript (v3)

> **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 i modelli di base disponibili.

```
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 API i dettagli, vedere ListFoundationModelsin AWS SDK for JavaScript APIReference.

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.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile 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.

SDKper JavaScript (v3)

> **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.

```
/**
```

```
 * @typedef {Object} Content
 * @property {string} text
 *
 * @typedef {Object} Usage
 * @property {number} input_tokens
 * @property {number} output_tokens
 *
 * @typedef {Object} ResponseBody
 * @property {Content[]} content
 * @property {Usage} usage
 */

import { fileURLToPath } from "url";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

const AWS_REGION = "us-east-1";

const MODEL_ID = "anthropic.claude-3-haiku-20240307-v1:0";
const PROMPT = "Hi. In a short paragraph, explain what you can do.";

const hello = async () => {
  console.log("=".repeat(35));
  console.log("Welcome to the Amazon Bedrock demo!");
  console.log("=".repeat(35));

  console.log("Model: Anthropic Claude 3 Haiku");
  console.log(`Prompt: ${PROMPT}\n`);
  console.log("Invoking model...\n");

  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: AWS_REGION });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [{ role: "user", content: [{ type: "text", text: PROMPT }] }],
  };

  // Invoke Claude with the payload and wait for the response.
  const apiResponse = await client.send(
```

```
    new InvokeModelCommand({
      contentType: "application/json",
      body: JSON.stringify(payload),
      modelId: MODEL_ID,
    }),
  );

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  const responses = responseBody.content;

  if (responses.length === 1) {
    console.log(`Response: ${responses[0].text}`);
  } else {
    console.log("Haiku returned multiple responses:");
    console.log(responses);
  }

  console.log(`\nNumber of input tokens:   ${responseBody.usage.input_tokens}`);
  console.log(`Number of output tokens: ${responseBody.usage.output_tokens}`);
};

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await hello();
}
```

- Per API i dettagli, vedi InvokeModel AWS SDK for JavaScriptAPIReference.

Argomenti

- Scenari
- AI21Laboratori Jurassic-2
- Testo Amazon Titan
- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

Scenari

Richiama più modelli di base su Amazon Bedrock

Il seguente esempio di codice mostra come preparare e inviare un prompt a una varietà di modelli in grandi lingue (LLMs) su Amazon Bedrock

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "url";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { FoundationModels } from "../config/foundation_models.js";

/**
 * @typedef {Object} ModelConfig
 * @property {Function} module
 * @property {Function} invoker
 * @property {string} modelId
 * @property {string} modelName
 */

const greeting = new ScenarioOutput(
  "greeting",
  "Welcome to the Amazon Bedrock Runtime client demo!",
  { header: true },
);

const selectModel = new ScenarioInput("model", "First, select a model:", {
  type: "select",
  choices: Object.values(FoundationModels).map((model) => ({
```

```
    name: model.modelName,
    value: model,
  })),
});

const enterPrompt = new ScenarioInput("prompt", "Now, enter your prompt:", {
  type: "input",
});

const printDetails = new ScenarioOutput(
  "print details",
  /**
   * @param {{ model: ModelConfig, prompt: string }} c
   */
  (c) => console.log(`Invoking ${c.model.modelName} with '${c.prompt}'...`),
  { slow: false },
);

const invokeModel = new ScenarioAction(
  "invoke model",
  /**
   * @param {{ model: ModelConfig, prompt: string, response: string }} c
   */
  async (c) => {
    const modelModule = await c.model.module();
    const invoker = c.model.invoker(modelModule);
    c.response = await invoker(c.prompt, c.model.modelId);
  },
);

const printResponse = new ScenarioOutput(
  "print response",
  /**
   * @param {{ response: string }} c
   */
  (c) => c.response,
  { slow: false },
);

const scenario = new Scenario("Amazon Bedrock Runtime Demo", [
  greeting,
  selectModel,
  enterPrompt,
  printDetails,
```

```
  invokeModel,
  printResponse,
]);

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  scenario.run();
}
```

- Per API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - InvokeModel
  - InvokeModelWithResponseStream

AI21Laboratori Jurassic-2

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a AI21 Labs Jurassic-2, utilizzando Converse di Bedrock. API

SDK JavaScript per (v3)

> **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.

Invia un messaggio di testo a AI21 Labs Jurassic-2, usando Converse di Bedrock. API

```
// Use the Conversation API to send a text message to AI21 Labs Jurassic-2.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });
```

```
// Set the model ID, e.g., Jurassic-2 Mid.
const modelId = "ai21.j2-mid-v1";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per i API dettagli, consulta Converse in Reference.AWS SDK for JavaScript API

InvokeModel

Il seguente esempio di codice mostra come inviare un messaggio di testo a AI21 Labs Jurassic-2, utilizzando il modello Invoke. API

SDKper (v3) 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.

Usa il modello Invoke API per inviare un messaggio di testo.

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Data
 * @property {string} text
 *
 * @typedef {Object} Completion
 * @property {Data} data
 *
 * @typedef {Object} ResponseBody
 * @property {Completion[]} completions
 */

/**
 * Invokes an AI21 Labs Jurassic-2 model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "ai21.j2-mid-
v1".
 */
export const invokeModel = async (prompt, modelId = "ai21.j2-mid-v1") => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
```

```
    prompt,
    maxTokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s).
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.completions[0].data.text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.JURASSIC2_MID.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- Per API i dettagli, vedere InvokeModelin AWS SDK for JavaScript APIReference.

Testo Amazon Titan

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo ad Amazon Titan Text utilizzando Converse di Bedrock. API

SDKper JavaScript (v3)

> **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.

Invia un messaggio di testo ad Amazon Titan Text utilizzando Converse di Bedrock. API

```
// Use the Conversation API to send a text message to Amazon Titan Text.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Titan Text Premier.
const modelId = "amazon.titan-text-premier-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
```

```
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per API i dettagli, consulta Converse in Reference.AWS SDK for JavaScript API

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo ad Amazon Titan Text utilizzando Converse di Bedrock API ed elaborare il flusso di risposta in tempo reale.

SDKper JavaScript (v3)

> **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.

Invia un messaggio di testo ad Amazon Titan Text utilizzando Converse di Bedrock API ed elabora il flusso di risposta in tempo reale.

```
// Use the Conversation API to send a text message to Amazon Titan Text.

import {
  BedrockRuntimeClient,
```

```
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Titan Text Premier.
const modelId = "amazon.titan-text-premier-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per i API dettagli, consulta Reference ConverseStream.AWS SDK for JavaScript API

InvokeModel

Il seguente esempio di codice mostra come inviare un messaggio di testo ad Amazon Titan Text utilizzando il modello Invoke. API

SDKper JavaScript (v3)

> **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.

Usa il modello Invoke API per inviare un messaggio di testo.

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {Object[]} results
 */

/**
 * Invokes an Amazon Titan Text generation model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "amazon.titan-text-express-v1".
 */
export const invokeModel = async (
  prompt,
  modelId = "amazon.titan-text-express-v1",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });
```

```
  // Prepare the payload for the model.
  const payload = {
    inputText: prompt,
    textGenerationConfig: {
      maxTokenCount: 4096,
      stopSequences: [],
      temperature: 0,
      topP: 1,
    },
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.results[0].outputText;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.TITAN_TEXT_G1_EXPRESS.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- Per API i dettagli, vedere InvokeModelin AWS SDK for JavaScript APIReference.

Anthropic Claude

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a Anthropic Claude, utilizzando Converse di Bedrock. API

SDKper (v3) 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.

Invia un messaggio di testo a Anthropic Claude, usando le Converse di Bedrock. API

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];
```

```
// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per API maggiori dettagli, consulta Converse in Reference.AWS SDK for JavaScript API

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo a Anthropic Claude utilizzando Converse di Bedrock API ed elaborare il flusso di risposta in tempo reale.

SDKper (v3) 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.

Invia un messaggio di testo a Anthropic Claude utilizzando Converse di Bedrock API ed elabora il flusso di risposta in tempo reale.

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
```

```
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per i API dettagli, consulta Reference. ConverseStreamAWS SDK for JavaScript API

InvokeModel

Il seguente esempio di codice mostra come inviare un messaggio di testo a Anthropic Claude, utilizzando il modello Invoke. API

SDKper JavaScript (v3)

> **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.

Usa il modello Invoke API per inviare un messaggio di testo.

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
```

```
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
```

```
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";
```

```
  // Decode and process the response stream
  for await (const item of apiResponse.body) {
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;

    if (chunk_type === "content_block_delta") {
      const text = chunk.delta.text;
      completeMessage = completeMessage + text;
      process.stdout.write(text);
    }
  }

  // Return the final response
  return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log("\n" + "-".repeat(53));
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- Per API i dettagli, vedere InvokeModelin AWS SDK for JavaScript APIReference.

InvokeModelWithResponseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo ai modelli Anthropic Claude, utilizzando il modello InvokeAPI, e stampare il flusso di risposta.

SDKper JavaScript (v3)

> **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.

Usa il modello Invoke API per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
```

```
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
};

/**
```

```
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;
```

```
    if (chunk_type === "content_block_delta") {
      const text = chunk.delta.text;
      completeMessage = completeMessage + text;
      process.stdout.write(text);
    }
  }

  // Return the final response
  return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log("\n" + "-".repeat(53));
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- Per API i dettagli, vedere InvokeModelWithResponseStreamin AWS SDK for JavaScript APIReference.

Cohere Command

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a Cohere Command, utilizzando Converse di Bedrock. API

SDKper JavaScript (v3)

> **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.

Invia un messaggio di testo a Cohere Command, usando Converse di Bedrock. API

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
```

```
  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per API i dettagli, consulta Converse in Reference.AWS SDK for JavaScript API

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo a Cohere Command, utilizzando Converse di Bedrock API ed elaborare il flusso di risposta in tempo reale.

SDKper JavaScript (v3)

> **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.

Invia un messaggio di testo a Cohere Command, utilizzando Converse di Bedrock API ed elabora il flusso di risposta in tempo reale.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";
```

```
// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per i API dettagli, consulta Reference ConverseStream.AWS SDK for JavaScript API

Meta Llama

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama utilizzando Converse di Bedrock. API

SDKper JavaScript (v3)

> **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.

Invia un messaggio di testo a Meta Llama utilizzando Converse di Bedrock. API

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
```

```
  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per API maggiori dettagli, consulta Converse in Reference.AWS SDK for JavaScript API

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama utilizzando Converse di Bedrock API ed elaborare il flusso di risposta in tempo reale.

SDKper JavaScript (v3)

> **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.

Invia un messaggio di testo a Meta Llama utilizzando Converse di Bedrock API ed elabora il flusso di risposta in tempo reale.

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";
```

```
// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per i API dettagli, consulta Reference [ConverseStream](#).AWS SDK for JavaScript API

InvokeModel: Llama 2

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama 2, utilizzando il modello Invoke. API

SDKper JavaScript (v3)

> **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.

Usa il modello Invoke API per inviare un messaggio di testo.

```
// Send a prompt to Meta Llama 2 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 2 Chat 13B.
const modelId = "meta.llama2-13b-chat-v1";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 2's prompt format.
const prompt = `<s>[INST] ${userMessage} [/INST]`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
```

```
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);

// Learn more about the Llama 2 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-2
```

- Per API i dettagli, vedere InvokeModelin AWS SDK for JavaScript APIReference.

InvokeModel: Llama 3

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama 3, utilizzando il modello Invoke. API

SDKper JavaScript (v3)

> **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.

Usa il modello Invoke API per inviare un messaggio di testo.

```
// Send a prompt to Meta Llama 3 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
```

```
// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);
```

```
// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- Per API i dettagli, vedere InvokeModelin AWS SDK for JavaScript APIReference.

InvokeModelWithResponseStream: Llama 2

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama 2, utilizzando il modello InvokeAPI, e stampare il flusso di risposta.

SDKper JavaScript (v3)

> **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.

Usa il modello Invoke API per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Send a prompt to Meta Llama 2 and print the response stream in real-time.

import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 2 Chat 13B.
const modelId = "meta.llama2-13b-chat-v1";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";
```

```
// Embed the message in Llama 2's prompt format.
const prompt = `<s>[INST] ${userMessage} [/INST]`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Extract and print the response stream in real-time.
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
  const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  if (chunk.generation) {
    process.stdout.write(chunk.generation);
  }
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- Per API i dettagli, vedere InvokeModelWithResponseStreamin AWS SDK for JavaScript APIReference.

InvokeModelWithResponseStream: Llama 3

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama 3, utilizzando il modello InvokeAPI, e stampare il flusso di risposta.

SDKper JavaScript (v3)

> **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.

Usa il modello Invoke API per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Send a prompt to Meta Llama 3 and print the response stream in real-time.

import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
```

```
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Extract and print the response stream in real-time.
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
  const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  if (chunk.generation) {
    process.stdout.write(chunk.generation);
  }
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-
tokens-used-with-meta-llama-3
```

- Per API i dettagli, vedere InvokeModelWithResponseStreamin AWS SDK for JavaScript APIReference.

Mistral AI

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a Mistral utilizzando Converse di Bedrock. API

SDKper (v3) 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.

Invia un messaggio di testo a Mistral utilizzando le Converse di Bedrock. API

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
```

```
  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per i API dettagli, consulta Converse in Reference.AWS SDK for JavaScript API

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo a Mistral, utilizzando Converse di Bedrock API ed elaborare il flusso di risposta in tempo reale.

SDKper (v3) 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.

Invia un messaggio di testo a Mistral utilizzando Converse di Bedrock API ed elabora il flusso di risposta in tempo reale.

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";
```

```
// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Per i API dettagli, consulta Reference. ConverseStreamAWS SDK for JavaScript API

InvokeModel

Il seguente esempio di codice mostra come inviare un messaggio di testo ai modelli Mistral, utilizzando Invoke Model. API

SDKper JavaScript (v3)

> **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.

Usa il modello Invoke API per inviare un messaggio di testo.

```
import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes a Mistral 7B Instruct model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "mistral.mistral-7b-instruct-v0:2".
 */
export const invokeModel = async (
  prompt,
  modelId = "mistral.mistral-7b-instruct-v0:2",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
```

```
  const instruction = `<s>[INST] ${prompt} [/INST]`;

  // Prepare the payload.
  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.outputs[0].text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time..."';
  const modelId = FoundationModels.MISTRAL_7B.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- Per API i dettagli, vedere InvokeModelin AWS SDK for JavaScript APIReference.

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. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile 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.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - GetAgent
  - ListAgents

Argomenti

- Azioni

Azioni

`CreateAgent`

Il seguente esempio di codice mostra come utilizzare`CreateAgent`.

SDKper JavaScript (v3)

> **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.

Creazione di un agente

```
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 API i dettagli, vedi CreateAgent AWS SDK for JavaScriptAPIReference.

`DeleteAgent`

Il seguente esempio di codice mostra come utilizzare`DeleteAgent`.

SDKper JavaScript (v3)

> **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.

Eliminare un agente.

```
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 API i dettagli, DeleteAgentconsulta AWS SDK for JavaScript APIReference.

`GetAgent`

Il seguente esempio di codice mostra come utilizzare`GetAgent`.

SDKper JavaScript (v3)

> **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.

Trovate un agente.

```
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 API i dettagli, vedi GetAgent AWS SDK for JavaScriptAPIReference.

`ListAgentActionGroups`

Il seguente esempio di codice mostra come utilizzare`ListAgentActionGroups`.

SDKper JavaScript (v3)

> **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 i gruppi d'azione di un agente.

```
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 API i dettagli, vedere ListAgentActionGroupsin AWS SDK for JavaScript APIReference.

`ListAgents`

Il seguente esempio di codice mostra come utilizzare`ListAgents`.

SDKper JavaScript (v3)

> **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 agenti appartenenti a un account.

```
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 API i dettagli, vedere ListAgentsin AWS SDK for JavaScript APIReference.

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. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`InvokeAgent`

Il seguente esempio di codice mostra come utilizzare`InvokeAgent`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 API i dettagli, vedi InvokeAgent AWS SDK for JavaScriptAPIReference.

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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`DeleteAlarms`

Il seguente esempio di codice mostra come utilizzare`DeleteAlarms`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere DeleteAlarmsin AWS SDK for JavaScript APIReference.

SDKper 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.

Importa i moduli SDK e client e chiama ilAPI.

```
// 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 API i dettagli, vedere DeleteAlarmsin AWS SDK for JavaScript APIReference.

`DescribeAlarmsForMetric`

Il seguente esempio di codice mostra come utilizzare`DescribeAlarmsForMetric`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere DescribeAlarmsForMetricin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi DescribeAlarmsForMetric AWS SDK for JavaScriptAPIReference.

`DisableAlarmActions`

Il seguente esempio di codice mostra come utilizzare`DisableAlarmActions`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere DisableAlarmActionsin AWS SDK for JavaScript APIReference.

SDKper 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.

Importa i moduli SDK e client e chiama ilAPI.

```
// 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 API i dettagli, vedere DisableAlarmActionsin AWS SDK for JavaScript APIReference.

`EnableAlarmActions`

Il seguente esempio di codice mostra come utilizzare`EnableAlarmActions`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere EnableAlarmActionsin AWS SDK for JavaScript APIReference.

SDKper 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.

Importa i moduli SDK e client e chiama ilAPI.

```
// 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 API i dettagli, vedere EnableAlarmActionsin AWS SDK for JavaScript APIReference.

`ListMetrics`

Il seguente esempio di codice mostra come utilizzare`ListMetrics`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere ListMetricsin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi ListMetrics AWS SDK for JavaScriptAPIReference.

`PutMetricAlarm`

Il seguente esempio di codice mostra come utilizzare`PutMetricAlarm`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere PutMetricAlarmin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutMetricAlarm AWS SDK for JavaScriptAPIReference.

`PutMetricData`

Il seguente esempio di codice mostra come utilizzare`PutMetricData`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere PutMetricDatain AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutMetricData AWS SDK for JavaScriptAPIReference.

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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel contesto degli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`PutEvents`

Il seguente esempio di codice mostra come utilizzare`PutEvents`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere PutEventsin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutEvents AWS SDK for JavaScriptAPIReference.

`PutRule`

Il seguente esempio di codice mostra come utilizzare`PutRule`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere PutRulein AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutRule AWS SDK for JavaScriptAPIReference.

`PutTargets`

Il seguente esempio di codice mostra come utilizzare`PutTargets`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere PutTargetsin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutTargets AWS SDK for JavaScriptAPIReference.

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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

- Scenari

Azioni

`CreateLogGroup`

Il seguente esempio di codice mostra come utilizzare`CreateLogGroup`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi CreateLogGroup AWS SDK for JavaScriptAPIReference.

`DeleteLogGroup`

Il seguente esempio di codice mostra come utilizzare`DeleteLogGroup`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteLogGroup AWS SDK for JavaScriptAPIReference.

`DeleteSubscriptionFilter`

Il seguente esempio di codice mostra come utilizzare`DeleteSubscriptionFilter`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteSubscriptionFilter AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DeleteSubscriptionFilter AWS SDK for JavaScriptAPIReference.

`DescribeLogGroups`

Il seguente esempio di codice mostra come utilizzare`DescribeLogGroups`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 API i dettagli, vedi DescribeLogGroups AWS SDK for JavaScriptAPIReference.

`DescribeSubscriptionFilters`

Il seguente esempio di codice mostra come utilizzare`DescribeSubscriptionFilters`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DescribeSubscriptionFilters AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DescribeSubscriptionFilters AWS SDK for JavaScriptAPIReference.

`GetQueryResults`

Il seguente esempio di codice mostra come utilizzare`GetQueryResults`.

SDKper JavaScript (v3)

> **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.

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
```

- Per API i dettagli, vedi GetQueryResults AWS SDK for JavaScriptAPIReference.

`PutSubscriptionFilter`

Il seguente esempio di codice mostra come utilizzare`PutSubscriptionFilter`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi PutSubscriptionFilter AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi PutSubscriptionFilter AWS SDK for JavaScriptAPIReference.

`StartLiveTail`

Il seguente esempio di codice mostra come utilizzare`StartLiveTail`.

SDKper JavaScript (v3)

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 API i dettagli, vedere StartLiveTailin AWS SDK for JavaScript API Reference.

`StartQuery`

Il seguente esempio di codice mostra come utilizzare`StartQuery`.

SDKper JavaScript (v3)

> **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.

```
/**
 * 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 API i dettagli, vedi StartQuery AWS SDK for JavaScriptAPIReference.

Scenari

Esegui una query di grandi dimensioni

Il seguente esempio di codice mostra come utilizzare CloudWatch Logs per interrogare più di 10.000 record.

SDKper JavaScript (v3)

> **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.

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]);
  }

  /**
   * Simple wrapper for the GetQueryResultsCommand.
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }

  /**
   * 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;
      }
    }
  }

  /**
   * 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;
    }
  }

  /**
   * 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 API i dettagli, consultate i seguenti argomenti in AWS SDK for JavaScript API Reference.
  - 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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`CreateProject`

Il seguente esempio di codice mostra come utilizzare`CreateProject`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi CreateProject AWS SDK for JavaScriptAPIReference.

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. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile 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.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
```

```
  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 API i dettagli, vedi ListUserPools AWS SDK for JavaScriptAPIReference.

Argomenti

- Azioni
- Scenari

Azioni

`AdminGetUser`

Il seguente esempio di codice mostra come utilizzare`AdminGetUser`.

SDKper JavaScript (v3)

> **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.

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
    Username: username,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi AdminGetUser AWS SDK for JavaScriptAPIReference.

`AdminInitiateAuth`

Il seguente esempio di codice mostra come utilizzare`AdminInitiateAuth`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi AdminInitiateAuth AWS SDK for JavaScriptAPIReference.

`AdminRespondToAuthChallenge`

Il seguente esempio di codice mostra come utilizzare`AdminRespondToAuthChallenge`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi AdminRespondToAuthChallenge AWS SDK for JavaScriptAPIReference.

`AssociateSoftwareToken`

Il seguente esempio di codice mostra come utilizzare`AssociateSoftwareToken`.

SDKper JavaScript (v3)

> **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.

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi AssociateSoftwareToken AWS SDK for JavaScriptAPIReference.

`ConfirmDevice`

Il seguente esempio di codice mostra come utilizzare`ConfirmDevice`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi ConfirmDevice AWS SDK for JavaScriptAPIReference.

`ConfirmSignUp`

Il seguente esempio di codice mostra come utilizzare`ConfirmSignUp`.

SDKper JavaScript (v3)

> **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.

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi ConfirmSignUp AWS SDK for JavaScriptAPIReference.

`InitiateAuth`

Il seguente esempio di codice mostra come utilizzare`InitiateAuth`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi InitiateAuth AWS SDK for JavaScriptAPIReference.

`ListUsers`

Il seguente esempio di codice mostra come utilizzare`ListUsers`.

SDKper JavaScript (v3)

> **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.

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});
```

```
  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi ListUsers AWS SDK for JavaScriptAPIReference.

`ResendConfirmationCode`

Il seguente esempio di codice mostra come utilizzare`ResendConfirmationCode`.

SDKper JavaScript (v3)

> **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.

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi ResendConfirmationCode AWS SDK for JavaScriptAPIReference.

`RespondToAuthChallenge`

Il seguente esempio di codice mostra come utilizzare`RespondToAuthChallenge`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi RespondToAuthChallenge AWS SDK for JavaScriptAPIReference.

`SignUp`

Il seguente esempio di codice mostra come utilizzare`SignUp`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi SignUp AWS SDK for JavaScriptAPIReference.

`VerifySoftwareToken`

Il seguente esempio di codice mostra come utilizzare`VerifySoftwareToken`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi VerifySoftwareToken AWS SDK for JavaScriptAPIReference.

Scenari

Registra un utente con un pool di utenti che richiede 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'MFAapplicazione all'utente.
- Accedi utilizzando una password e un MFA codice.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript API Reference.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

Esempi di Amazon Comprehend 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 Comprehend.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Scenari

Scenari

Creazione di un'app in streaming Amazon Transcribe

L'esempio di codice seguente mostra come creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia tramite e-mail i risultati.

SDKper JavaScript (v3)

Mostra come usare Amazon Transcribe per creare un'app che registra, trascrive e traduce audio dal vivo in tempo reale e invia tramite e-mail i risultati utilizzando Amazon Simple Email Service (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 Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Creazione di un chatbot Amazon Lex

Il seguente esempio di codice mostra come creare un chatbot per coinvolgere i visitatori del tuo sito web.

SDKper JavaScript (v3)

Mostra come utilizzare Amazon Lex API 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

Crea un'applicazione per analizzare il feedback dei clienti

L'esempio di codice seguente mostra come creare un'applicazione che analizza le schede dei commenti dei clienti, le traduce dalla loro lingua originale, ne determina il sentiment e genera un file audio dal testo tradotto.

SDKper 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

Esempi di Amazon DocumentDB 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 DocumentDB.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Esempi serverless

Esempi serverless

Richiama una funzione Lambda da un trigger di Amazon DocumentDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso di modifiche di DocumentDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

SDKper (v3) JavaScript

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Amazon DocumentDB con Lambda utilizzando. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

Consumo di un evento Amazon DocumentDB con Lambda utilizzando TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

Esempi di DynamoDB SDK con 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 basi sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto nei relativi scenari.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui 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.

SDKper JavaScript (v3)

> **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.

Per maggiori dettagli sull'utilizzo di DynamoDB AWS SDK for JavaScript in, consulta Programmazione di DynamoDB con. JavaScript

```
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 API dettagli, consulta la sezione Reference. ListTablesAWS SDK for JavaScript API

Argomenti

- Nozioni di base
- Azioni
- Scenari
- Esempi serverless

Nozioni di base

Scopri le nozioni di base

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 sulla tabella a partire da un JSON file 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.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Azioni

`BatchExecuteStatement`

Il seguente esempio di codice mostra come utilizzare`BatchExecuteStatement`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi [BatchExecuteStatement AWS SDK for JavaScript](#)APIReference.

`BatchGetItem`

Il seguente esempio di codice mostra come utilizzare`BatchGetItem`.

SDKper JavaScript (v3)

> **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.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per API i dettagli vedi 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 API i dettagli, vedere BatchGetItemin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi BatchGetItem AWS SDK for JavaScriptAPIReference.

`BatchWriteItem`

Il seguente esempio di codice mostra come utilizzare`BatchWriteItem`.

SDKper JavaScript (v3)

> **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.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per API i dettagli vedi 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 API i dettagli, vedere BatchWriteItemin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi BatchWriteItem AWS SDK for JavaScriptAPIReference.

`CreateTable`

Il seguente esempio di codice mostra come utilizzare`CreateTable`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi CreateTable AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi CreateTable AWS SDK for JavaScriptAPIReference.

`DeleteItem`

Il seguente esempio di codice mostra come utilizzare`DeleteItem`.

SDKper JavaScript (v3)

> **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.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per API i dettagli vedi 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 API i dettagli, vedere DeleteItemin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi DeleteItem AWS SDK for JavaScriptAPIReference.

`DeleteTable`

Il seguente esempio di codice mostra come utilizzare`DeleteTable`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteTable AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DeleteTable AWS SDK for JavaScriptAPIReference.

`DescribeTable`

Il seguente esempio di codice mostra come utilizzare`DescribeTable`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DescribeTable AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DescribeTable AWS SDK for JavaScriptAPIReference.

`DescribeTimeToLive`

Il seguente esempio di codice mostra come utilizzare`DescribeTimeToLive`.

SDKper JavaScript (v3)

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

const describeTableTTL = async (tableName, region) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    try {
        const ttlDescription = await client.send(new
 DescribeTimeToLiveCommand({ TableName: tableName }));

        if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED') {
            console.log("TTL is enabled for table %s.", tableName);
        } else {
            console.log("TTL is not enabled for table %s.", tableName);
        }

        return ttlDescription;
    } catch (e) {
        console.error(`Error describing table: ${e}`);
        throw e;
    }
}

// enter table name and change region if desired.
describeTableTTL('your-table-name', 'us-east-1');
```

- Per API i dettagli, vedere DescribeTimeToLivein AWS SDK for JavaScript APIReference.

`ExecuteStatement`

Il seguente esempio di codice mostra come utilizzare`ExecuteStatement`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi ExecuteStatement AWS SDK for JavaScriptAPIReference.

`GetItem`

Il seguente esempio di codice mostra come utilizzare`GetItem`.

SDKper JavaScript (v3)

> **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.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per API i dettagli vedi 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 API i dettagli, vedere GetItemin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi GetItem AWS SDK for JavaScriptAPIReference.

`ListTables`

Il seguente esempio di codice mostra come utilizzare`ListTables`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListTables AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi ListTables AWS SDK for JavaScriptAPIReference.

`PutItem`

Il seguente esempio di codice mostra come utilizzare`PutItem`.

SDKper JavaScript (v3)

> **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.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per API i dettagli vedi 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 API i dettagli, vedere PutItemin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutItem AWS SDK for JavaScriptAPIReference.

`Query`

Il seguente esempio di codice mostra come utilizzare`Query`.

SDKper JavaScript (v3)

> **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.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per API i dettagli vedi 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 API i dettagli, vedere Query in AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi Query in AWS SDK for JavaScript APIReference.

`Scan`

Il seguente esempio di codice mostra come utilizzare`Scan`.

SDKper JavaScript (v3)

> **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.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per API i dettagli vedi 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 API i dettagli, consulta Scan in AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, consulta Scan in AWS SDK for JavaScript APIReference.

`UpdateItem`

Il seguente esempio di codice mostra come utilizzare`UpdateItem`.

SDKper JavaScript (v3)

> **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.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per API i dettagli vedi 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 API i dettagli, vedere UpdateItemin AWS SDK for JavaScript APIReference.

`UpdateTimeToLive`

Il seguente esempio di codice mostra come utilizzare`UpdateTimeToLive`.

SDKper JavaScript (v3)

Abilita TTL su una tabella DynamoDB esistente.

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

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
```

```
        throw e;
    }
};

// call with your own values
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Disabilita TTL su una tabella DynamoDB esistente.

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

const disableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL disabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to disable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Per API i dettagli, consulta la sezione UpdateTimeToLiveReference AWS SDK for JavaScript API.

Scenari

Costruisci un'app per inviare dati a una tabella DynamoDB

Il seguente esempio di codice mostra come creare un'applicazione che invia dati a una tabella Amazon DynamoDB e ti avvisa quando un utente aggiorna la tabella.

SDKper (v3) JavaScript

Questo esempio mostra come creare un'app che consenta agli utenti di inviare dati a una tabella Amazon DynamoDB e inviare 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, consulta 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

Aggiorna in modo condizionale un articolo TTL

Il seguente esempio di codice mostra come aggiornare in modo condizionale un elemento. TTL

SDKper JavaScript (v3)

Aggiorna TTL un elemento DynamoDB esistente in una tabella, con una condizione.

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

const updateDynamoDBItem = async (tableName, region, partitionKey, sortKey,
 newAttribute) => {
```

```
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Enter your values here
updateDynamoDBItem('your-table-name', "us-east-1",'your-partition-key-value', 'your-
sort-key-value', 'your-new-attribute-value');
```

- Per API i dettagli, consulta la sezione UpdateItemReference AWS SDK for JavaScript API.

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDKper 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, vedi l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- APIGateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Creazione di un'applicazione Web per tracciare i dati DynamoDB

Il seguente esempio di codice mostra come creare un'applicazione Web che tiene traccia degli elementi di lavoro in una tabella Amazon DynamoDB e utilizza Amazon Simple Email Service (SESAmazon) per inviare report.

SDKper JavaScript (v3)

Mostra come utilizzare Amazon DynamoDB per creare un'applicazione Web API dinamica che tenga traccia dei dati di lavoro di DynamoDB.

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 SES

Crea un articolo con un TTL

Il seguente esempio di codice mostra come creare un elemento conTTL.

SDKper JavaScript (v3)

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

function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 * 1000) /
 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
```

```
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, ex);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// use your own values
createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Per API i dettagli, vedere PutItemin AWS SDK for JavaScript APIReference.

Rileva PPE nelle immagini

Il seguente esempio di codice mostra come creare un'app che utilizza Amazon Rekognition per rilevare i dispositivi di protezione PPE individuale () nelle immagini.

SDKper JavaScript (v3)

Mostra come usare Amazon Rekognition AWS SDK for JavaScript con per creare un'applicazione per rilevare dispositivi di protezione PPE individuale () nelle immagini che si trovano in un bucket Amazon Simple Storage Service (Amazon S3). L'app salva i risultati in una tabella Amazon DynamoDB e invia all'amministratore una notifica e-mail con i risultati utilizzando Amazon Simple Email Service (Amazon). SES

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per PPE usare Amazon Rekognition.
- Verifica un indirizzo email per AmazonSES.
- Aggiornare una tabella DynamoDB con i risultati.
- Invia una notifica via e-mail tramite AmazonSES.

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 Rekognition
- Amazon S3
- Amazon SES

Richiamo a una funzione Lambda da un browser

Il seguente esempio di codice mostra come richiamare una AWS Lambda funzione da un browser.

SDKper 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

SDKper 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

Esecuzione di una query su una tabella mediante batch di istruzioni PartiQL

L'esempio di codice seguente mostra come:

- Ottieni un batch di elementi eseguendo più SELECT istruzioni.
- Aggiungi un batch di elementi eseguendo più INSERT istruzioni.
- Aggiorna un batch di elementi eseguendo più UPDATE istruzioni.
- Eliminare un batch di elementi eseguendo più DELETE istruzioni.

SDKper JavaScript (v3)

> **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.

Eseguire le istruzioni PartiQL in batch.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

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

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
```

```
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
```

```
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
```

```
  });
  await docClient.send(addItemsStatementCommand);
  log(`Cities inserted.`);

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
```

```
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log(`Updated cities.`);

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Per API i dettagli, vedi [BatchExecuteStatement AWS SDK for JavaScript](#)APIReference.

Esecuzione di una query mediante PartiQL

L'esempio di codice seguente mostra come:

- Ottieni un elemento eseguendo una SELECT dichiarazione.
- Aggiungi un elemento eseguendo un'INSERTistruzione.
- Aggiorna un elemento eseguendo un'UPDATEistruzione.
- Eliminare un elemento eseguendo un'DELETEistruzione.

SDKper JavaScript (v3)

> **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.

Eseguire le singole istruzioni PartiQL.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

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

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
```

```
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
```

```
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log(`Coffee inserted.`);
```

```
  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log(`Updated coffee`);

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
```

```
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Per API i dettagli, vedi ExecuteStatement AWS SDK for JavaScriptAPIReference.

Richiesta di TTL articoli

Il seguente esempio di codice mostra come eseguire una query per TTL gli elementi.

SDKper JavaScript (v3)

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

async function queryDynamoDBItems(tableName, region, primaryKey) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };
```

```
    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

//enter your own values here
queryDynamoDBItems('your-table-name', 'your-partition-key-value');
```

- Per API i dettagli, vedere Query in AWS SDK for JavaScript APIReference.

Aggiorna un elemento TTL

Il seguente esempio di codice mostra come aggiornare un articoloTTL.

SDKper JavaScript (v3)

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

async function updateDynamoDBItem(tableName, region, partitionKey, sortKey) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
```

```
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

//enter your values here
updateDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Per API i dettagli, vedere UpdateItemin AWS SDK for JavaScript APIReference.

Esempi serverless

Richiama una funzione Lambda da un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DynamoDB e registra il contenuto del record.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento DynamoDB con Lambda utilizzando. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consumo di un evento DynamoDB con Lambda utilizzando. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Segnalazione degli errori degli elementi batch per le funzioni Lambda con un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando Lambda. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando Lambda. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;
```

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

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

EC2Esempi di Amazon 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 AmazonEC2.

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto nei relativi scenari.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Ciao Amazon EC2

I seguenti esempi di codice mostrano come iniziare a usare AmazonEC2.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  const client = new EC2Client();
  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);
  }
};

// Call function if run directly.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Per API i dettagli, vedi DescribeSecurityGroups AWS SDK for JavaScriptAPIReference.

Argomenti

Nozioni di base

Scopri le nozioni di base

L'esempio di codice seguente mostra come:

- Creare una coppia di chiavi e un gruppo di sicurezza.
- Seleziona un'Amazon Machine Image (AMI) e un tipo di istanza compatibile, quindi crea un'istanza.
- Arrestare e riavviare l'istanza.
- Associazione di un indirizzo IP elastico all'istanza
- Connettiti alla tua istanza conSSH, quindi pulisci le risorse.

SDKper JavaScript (v3)

> **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.

Questo file contiene un elenco di azioni comuni utilizzate conEC2. I passaggi sono costruiti con un framework Scenario che semplifica l'esecuzione di un esempio interattivo. Per il contesto completo, visita il GitHub repository.

```
import { tmpdir } from "node:os";
import { writeFile, mkdtemp, rm } from "node:fs/promises";
import { join } from "node:path";
import { get } from "node:http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
```

```
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
  DeleteSecurityGroupCommand,
  DisassociateAddressCommand,
  paginateDescribeImages,
  paginateDescribeInstances,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

/**
 * @typedef {{
 *   ec2Client: import('@aws-sdk/client-ec2').EC2Client,
 *   errors: Error[],
 *   keyPairId?: string,
 *   tmpDirectory?: string,
 *   securityGroupId?: string,
 *   ipAddress?: string,
 *   images?: import('@aws-sdk/client-ec2').Image[],
 *   image?: import('@aws-sdk/client-ec2').Image,
 *   instanceTypes?: import('@aws-sdk/client-ec2').InstanceTypeInfo[],
 *   instanceId?: string,
 *   instanceIpAddress?: string,
 *   allocationId?: string,
 *   allocatedIpAddress?: string,
 *   associationId?: string,
 * }} State
 */
```

```
/**
 * A skip function provided to the `skipWhen` of a Step when you want
 * to ignore that step if any errors have occurred.
 * @param {State} state
 */
const skipWhenErrors = (state) => state.errors.length > 0;

const MAX_WAITER_TIME_IN_SECONDS = 60 * 8;

export const confirm = new ScenarioInput("confirmContinue", "Continue?", {
  type: "confirm",
  skipWhen: skipWhenErrors,
});

export const exitOnNoConfirm = new ScenarioAction(
  `exitOnConfirmContinueFalse`,
  (/** @type { { earlyExit: boolean } & Record<string, any>} */ state) => {
    if (!state[confirm.name]) {
      state.earlyExit = true;
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

export const greeting = new ScenarioOutput(
  "greeting",
  `Welcome to the Amazon EC2 basic usage scenario.
Before you launch an instances, you'll need to provide a few things:
 • A key pair - This is for SSH access to your EC2 instance. You only need to
 provide the name.
 • A security group - This is used for configuring access to your instance. Again,
 only the name is needed.
 • An IP address - Your public IP address will be fetched.
 • An Amazon Machine Image (AMI)
 • A compatible instance type`,
  { header: true, preformatted: true, skipWhen: skipWhenErrors },
);

export const provideKeyPairName = new ScenarioInput(
  "keyPairName",
  "Provide a name for a new key pair.",
```

```
  { type: "input", default: "ec2-example-key-pair", skipWhen: skipWhenErrors },
);

export const createKeyPair = new ScenarioAction(
  "createKeyPair",
  async (/** @type {State} */ state) => {
    try {
      // Create a key pair in Amazon EC2.
      const { KeyMaterial, KeyPairId } = await state.ec2Client.send(
        // A unique name for the key pair. Up to 255 ASCII characters.
        new CreateKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
      );

      state.keyPairId = KeyPairId;

      // Save the private key in a temporary location.
      state.tmpDirectory = await mkdtemp(join(tmpdir(), "ec2-scenario-tmp"));
      await writeFile(
        `${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem`,
        KeyMaterial,
        {
          mode: 0o400,
        },
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidKeyPair.Duplicate"
      ) {
        caught.message = `${caught.message}. Try another key name.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logKeyPair = new ScenarioOutput(
  "logKeyPair",
  (/** @type {State} */ state) =>
    `Created the key pair ${state[provideKeyPairName.name]}.`,
  { skipWhen: skipWhenErrors },
);
```

```
export const confirmDeleteKeyPair = new ScenarioInput(
  "confirmDeleteKeyPair",
  "Do you want to delete the key pair?",
  {
    type: "confirm",
    // Don't do anything when a key pair was never created.
    skipWhen: (/** @type {State} */ state) => !state.keyPairId,
  },
);

export const maybeDeleteKeyPair = new ScenarioAction(
  "deleteKeyPair",
  async (/** @type {State} */ state) => {
    try {
      // Delete a key pair by name from EC2
      await state.ec2Client.send(
        new DeleteKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        // Occurs when a required parameter (e.g. KeyName) is undefined.
        caught.name === "MissingParameter"
      ) {
        caught.message = `${caught.message}. Did you provide the required value?`;
      }
      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no key pair to delete or the user chooses
    // to keep it.
    skipWhen: (/** @type {State} */ state) =>
      !state.keyPairId || !state[confirmDeleteKeyPair.name],
  },
);

export const provideSecurityGroupName = new ScenarioInput(
  "securityGroupName",
  "Provide a name for a new security group.",
  { type: "input", default: "ec2-scenario-sg", skipWhen: skipWhenErrors },
);
```

```
export const createSecurityGroup = new ScenarioAction(
  "createSecurityGroup",
  async (/** @type {State} */ state) => {
    try {
      // Create a new security group that will be used to configure ingress/egress
 for
      // an EC2 instance.
      const { GroupId } = await state.ec2Client.send(
        new CreateSecurityGroupCommand({
          GroupName: state[provideSecurityGroupName.name],
          Description: "A security group for the Amazon EC2 example.",
        }),
      );
      state.securityGroupId = GroupId;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidGroup.Duplicate") {
        caught.message = `${caught.message}. Please provide a different name for
 your security group.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSecurityGroup = new ScenarioOutput(
  "logSecurityGroup",
  (/** @type {State} */ state) =>
    `Created the security group ${state.securityGroupId}.`,
  { skipWhen: skipWhenErrors },
);

export const confirmDeleteSecurityGroup = new ScenarioInput(
  "confirmDeleteSecurityGroup",
  "Do you want to delete the security group?",
  {
    type: "confirm",
    // Don't do anything when a security group was never created.
    skipWhen: (/** @type {State} */ state) => !state.securityGroupId,
  },
);

export const maybeDeleteSecurityGroup = new ScenarioAction(
```

```
  "deleteSecurityGroup",
  async (/** @type {State} */ state) => {
    try {
      // Delete the security group if the 'skipWhen' condition below is not met.
      await state.ec2Client.send(
        new DeleteSecurityGroupCommand({
          GroupId: state.securityGroupId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidGroupId.Malformed"
      ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
      }
      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no security group to delete
    // or the user chooses to keep it.
    skipWhen: (/** @type {State} */ state) =>
      !state.securityGroupId || !state[confirmDeleteSecurityGroup.name],
  },
);

export const authorizeSecurityGroupIngress = new ScenarioAction(
  "authorizeSecurity",
  async (/** @type {State} */ state) => {
    try {
      // Get the public IP address of the machine running this example.
      const ipAddress = await 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);
        });
      });
      state[`ipAddress`] = ipAddress;
      // Allow ingress from the IP address above to the security group.
      // This will allow you to SSH into the EC2 instance.
```

```
      const command = new AuthorizeSecurityGroupIngressCommand({
        GroupId: state.securityGroupId,
        IpPermissions: [
          {
            IpProtocol: "tcp",
            FromPort: 22,
            ToPort: 22,
            IpRanges: [{ CidrIp: `${ipAddress}/32` }],
          },
        ],
      });

      await state.ec2Client.send(command);
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidGroupId.Malformed"
      ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSecurityGroupIngress = new ScenarioOutput(
  "logSecurityGroupIngress",
  (/** @type {State} */ state) =>
    `Allowed SSH access from your public IP: ${state.ipAddress}.`,
  { skipWhen: skipWhenErrors },
);

export const getImages = new ScenarioAction(
  "images",
  async (/** @type {State} */ state) => {
    const AMIs = [];
    // Some AWS services publish information about common artifacts as AWS Systems
 Manager (SSM)
    // public parameters. For example, the Amazon Elastic Compute Cloud (Amazon EC2)
    // service publishes information about Amazon Machine Images (AMIs) as public
 parameters.
```

```
  // Create the paginator for getting images. Actions that return multiple pages
of
  // results have paginators to simplify those calls.
  const getParametersByPathPaginator = paginateGetParametersByPath(
    {
      // Not storing this client in state since it's only used once.
      client: new SSMClient({}),
    },
    {
      // The path to the public list of the latest amazon-linux instances.
      Path: "/aws/service/ami-amazon-linux-latest",
    },
  );

  try {
    for await (const page of getParametersByPathPaginator) {
      page.Parameters.forEach((param) => {
        // Filter by Amazon Linux 2
        if (param.Name.includes("amzn2")) {
          AMIs.push(param.Value);
        }
      });
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidFilterValue") {
      caught.message = `${caught.message} Please provide a valid filter value for
paginateGetParametersByPath.`;
    }
    state.errors.push(caught);
    return;
  }

  const imageDetails = [];
  const describeImagesPaginator = paginateDescribeImages(
    { client: state.ec2Client },
    // The images found from the call to SSM.
    { ImageIds: AMIs },
  );

  try {
    // Get more details for the images found above.
    for await (const page of describeImagesPaginator) {
      imageDetails.push(...(page.Images || []));
    }
```

```
      // Store the image details for later use.
      state["images"] = imageDetails;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidAMIID.NotFound") {
        caught.message = `${caught.message}. Please provide a valid image id.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const provideImage = new ScenarioInput(
  "image",
  "Select one of the following images.",
  {
    type: "select",
    choices: (/** @type { State } */ state) =>
      state.images.map((image) => ({
        name: `${image.ImageId} - ${image.Description}`,
        value: image,
      })),
    default: (/** @type { State } */ state) => state.images[0],
    skipWhen: skipWhenErrors,
  },
);

export const getCompatibleInstanceTypes = new ScenarioAction(
  "getCompatibleInstanceTypes",
  async (/** @type {State} */ state) => {
    // Get more details about instance types that match the architecture of
    // the provided image.
    const paginator = paginateDescribeInstanceTypes(
      { client: state.ec2Client, pageSize: 25 },
      {
        Filters: [
          {
            Name: "processor-info.supported-architecture",
            // The value selected from provideImage()
            Values: [state.image.Architecture],
          },
          // Filter for smaller, less expensive, types.
```

```
          { Name: "instance-type", Values: ["*.micro", "*.small"] },
        ],
      },
    );

    const instanceTypes = [];

    try {
      for await (const page of paginator) {
        if (page.InstanceTypes.length) {
          instanceTypes.push(...(page.InstanceTypes || []));
        }
      }

      if (!instanceTypes.length) {
        state.errors.push(
          "No instance types matched the instance type filters.",
        );
      }
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check the provided values and
 try again.`;
      }

      state.errors.push(caught);
    }

    state["instanceTypes"] = instanceTypes;
  },
  { skipWhen: skipWhenErrors },
);

export const provideInstanceType = new ScenarioInput(
  "instanceType",
  "Select an instance type.",
  {
    choices: (/** @type {State} */ state) =>
      state.instanceTypes.map((instanceType) => ({
        name: `${instanceType.InstanceType} - Memory:
${instanceType.MemoryInfo.SizeInMiB}`,
        value: instanceType.InstanceType,
      })),
    type: "select",
```

```
    default: (/** @type {State} */ state) =>
      state.instanceTypes[0].InstanceType,
    skipWhen: skipWhenErrors,
  },
);

export const runInstance = new ScenarioAction(
  "runInstance",
  async (/** @type { State } */ state) => {
    const { Instances } = await state.ec2Client.send(
      new RunInstancesCommand({
        KeyName: state[provideKeyPairName.name],
        SecurityGroupIds: [state.securityGroupId],
        ImageId: state.image.ImageId,
        InstanceType: state[provideInstanceType.name],
        // Availability Zones have capacity limitations that may impact your ability
 to launch instances.
        // The `RunInstances` operation will only succeed if it can allocate at
 least the `MinCount` of instances.
        // However, EC2 will attempt to launch up to the `MaxCount` of instances,
 even if the full request cannot be satisfied.
        // If you need a specific number of instances, use `MinCount` and `MaxCount`
 set to the same value.
        // If you want to launch up to a certain number of instances, use `MaxCount`
 and let EC2 provision as many as possible.
        // If you require a minimum number of instances, but do not want to exceed a
 maximum, use both `MinCount` and `MaxCount`.
        MinCount: 1,
        MaxCount: 1,
      }),
    );

    state.instanceId = Instances[0].InstanceId;

    try {
      // Poll `DescribeInstanceStatus` until status is "ok".
      await waitUntilInstanceStatusOk(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [Instances[0].InstanceId] },
      );
    } catch (caught) {
```

```
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logRunInstance = new ScenarioOutput(
  "logRunInstance",
  "The next step is to run your EC2 instance for the first time. This can take a few
 minutes.",
  { header: true, skipWhen: skipWhenErrors },
);

export const describeInstance = new ScenarioAction(
  "describeInstance",
  async (/** @type { State } */ state) => {
    /** @type { import("@aws-sdk/client-ec2").Instance[] } */
    const instances = [];

    try {
      const paginator = paginateDescribeInstances(
        {
          client: state.ec2Client,
        },
        {
          // Only get our created instance.
          InstanceIds: [state.instanceId],
        },
      );

      for await (const page of paginator) {
        for (const reservation of page.Reservations) {
          instances.push(...reservation.Instances);
        }
      }
      if (instances.length !== 1) {
        throw new Error(`Instance ${state.instanceId} not found.`);
      }
```

```
      // The only info we need is the IP address for SSH purposes.
      state.instanceIpAddress = instances[0].PublicIpAddress;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check provided values and try
 again.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSSHConnectionInfo = new ScenarioOutput(
  "logSSHConnectionInfo",
  (/** @type { State } */ state) =>
    `You can now SSH into your instance using the following command:
ssh -i ${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem ec2-user@
${state.instanceIpAddress}`,
  { preformatted: true, skipWhen: skipWhenErrors },
);

export const logStopInstance = new ScenarioOutput(
  "logStopInstance",
  "Stopping your EC2 instance.",
  { skipWhen: skipWhenErrors },
);

export const stopInstance = new ScenarioAction(
  "stopInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StopInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStopped(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
```

```
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  // Don't try to stop an instance that doesn't exist.
  { skipWhen: (/** @type { State } */ state) => !state.instanceId },
);

export const logIpAddressBehavior = new ScenarioOutput(
  "logIpAddressBehavior",
  [
    "When you run an instance, by default it's assigned an IP address.",
    "That IP address is not static. It will change every time the instance is
 restarted.",
    "The next step is to stop and restart your instance to demonstrate this
 behavior.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const logStartInstance = new ScenarioOutput(
  "logStartInstance",
  (/** @type { State } */ state) => `Starting instance ${state.instanceId}`,
  { skipWhen: skipWhenErrors },
);

export const startInstance = new ScenarioAction(
  "startInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StartInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStatusOk(
```

```
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logIpAllocation = new ScenarioOutput(
  "logIpAllocation",
  [
    "It is possible to have a static IP address.",
    "To demonstrate this, an IP will be allocated and associated to your EC2
 instance.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const allocateIp = new ScenarioAction(
  "allocateIp",
  async (/** @type { State } */ state) => {
    try {
      // An Elastic IP address is allocated to your AWS account, and is yours until
 you release it.
      const { AllocationId, PublicIp } = await state.ec2Client.send(
        new AllocateAddressCommand({}),
      );
      state.allocationId = AllocationId;
      state.allocatedIpAddress = PublicIp;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        caught.message = `${caught.message}. Did you provide these values?`;
      }
      state.errors.push(caught);
```

```
    }
  },
  { skipWhen: skipWhenErrors },
);

export const associateIp = new ScenarioAction(
  "associateIp",
  async (/** @type { State } */ state) => {
    try {
      // Associate an allocated IP address to an EC2 instance. An IP address can be
 allocated
      // with the AllocateAddress action.
      const { AssociationId } = await state.ec2Client.send(
        new AssociateAddressCommand({
          AllocationId: state.allocationId,
          InstanceId: state.instanceId,
        }),
      );
      state.associationId = AssociationId;
      // Update the IP address that is being tracked to match
      // the one just associated.
      state.instanceIpAddress = state.allocatedIpAddress;
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Did you provide the ID of a valid
 Elastic IP address AllocationId?`;
      }
      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logStaticIpProof = new ScenarioOutput(
  "logStaticIpProof",
  "The IP address should remain the same even after stopping and starting the
 instance.",
  { header: true, skipWhen: skipWhenErrors },
);

export const logCleanUp = new ScenarioOutput(
```

```
  "logCleanUp",
  "That's it! You can choose to clean up the resources now, or clean them up on your
 own later.",
  { header: true, skipWhen: skipWhenErrors },
);

export const confirmDisassociateAddress = new ScenarioInput(
  "confirmDisassociateAddress",
  "Do you want to disassociate and release the static IP address created earlier?",
  {
    type: "confirm",
    skipWhen: (/** @type { State } */ state) => !state.associationId,
  },
);

export const maybeDisassociateAddress = new ScenarioAction(
  "maybeDisassociateAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new DisassociateAddressCommand({
          AssociationId: state.associationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAssociationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid association
 ID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.associationId,
  },
);

export const maybeReleaseAddress = new ScenarioAction(
  "maybeReleaseAddress",
  async (/** @type { State } */ state) => {
```

```
    try {
      await state.ec2Client.send(
        new ReleaseAddressCommand({
          AllocationId: state.allocationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid AllocationID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.allocationId,
  },
);

export const confirmTerminateInstance = new ScenarioInput(
  "confirmTerminateInstance",
  "Do you want to terminate the instance?",
  // Don't do anything when an instance was never run.
  {
    skipWhen: (/** @type { State } */ state) => !state.instanceId,
    type: "confirm",
  },
);

export const maybeTerminateInstance = new ScenarioAction(
  "terminateInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new TerminateInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );
      await waitUntilInstanceTerminated(
        { client: state.ec2Client },
        { InstanceIds: [state.instanceId] },
```

```
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no instance to terminate or the
    // use chooses not to terminate.
    skipWhen: (/** @type { State } */ state) =>
      !state.instanceId || !state[confirmTerminateInstance.name],
  },
);

export const deleteTemporaryDirectory = new ScenarioAction(
  "deleteTemporaryDirectory",
  async (/** @type { State } */ state) => {
    try {
      await rm(state.tmpDirectory, { recursive: true });
    } catch (caught) {
      state.errors.push(caught);
    }
  },
);

export const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State}*/ state) => {
    const errorList = state.errors
      .map((err) => `• ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    preformatted: true,
    header: true,
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);
```

- Per API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIReference.
  - AllocateAddress
  - AssociateAddress
  - AuthorizeSecurityGroupIngress
  - CreateKeyPair
  - CreateSecurityGroup
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DescribeImages
  - DescribeInstanceTypes
  - DescribeInstances
  - DescribeKeyPairs
  - DescribeSecurityGroups
  - DisassociateAddress
  - ReleaseAddress
  - RunInstances
  - StartInstances
  - StopInstances
  - TerminateInstances
  - UnmonitorInstances

Azioni

`AllocateAddress`

Il seguente esempio di codice mostra come utilizzare`AllocateAddress`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { AllocateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Allocates an Elastic IP address to your AWS account.
 */
export const main = async () => {
  const client = new EC2Client({});
  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: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2.
 Look under Network & Security > Elastic IPs",
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
import { fileURLToPath } from "url";
// Call function if run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Per API i dettagli, vedi AllocateAddress AWS SDK for JavaScriptAPIReference.

`AssociateAddress`

Il seguente esempio di codice mostra come utilizzare`AssociateAddress`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { AssociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Associates an Elastic IP address, or carrier IP address (for instances that are
 in subnets in Wavelength Zones)
 * with an instance or a network interface.
 * @param {{ instanceId: string, allocationId: string }} options
 */
export const main = async ({ instanceId, allocationId }) => {
  const client = new EC2Client({});
  const command = new AssociateAddressCommand({
    // You need to allocate an Elastic IP address before associating it with an
 instance.
    // You can do that with the AllocateAddressCommand.
    AllocationId: allocationId,
    // You need to create an EC2 instance before an IP address can be associated
 with it.
    // You can do that with the RunInstancesCommand.
    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 (caught) {
    if (
      caught instanceof Error &&
```

```
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Did you provide the ID of a valid Elastic IP address
 AllocationId?`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi AssociateAddress AWS SDK for JavaScriptAPIReference.

`AuthorizeSecurityGroupIngress`

Il seguente esempio di codice mostra come utilizzare`AuthorizeSecurityGroupIngress`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  AuthorizeSecurityGroupIngressCommand,
  EC2Client,
} from "@aws-sdk/client-ec2";

/**
 * Adds the specified inbound (ingress) rules to a security group.
 * @param {{ groupId: string, ipAddress: string }} options
 */
export const main = async ({ groupId, ipAddress }) => {
  const client = new EC2Client({});
  const command = new AuthorizeSecurityGroupIngressCommand({
    // Use a group ID from the AWS console or
    // the DescribeSecurityGroupsCommand.
    GroupId: groupId,
```

```
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        // 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: `${ipAddress}/32` }],
      },
    ],
  });

  try {
    const { SecurityGroupRules } = await client.send(command);
    console.log(JSON.stringify(SecurityGroupRules, null, 2));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi AuthorizeSecurityGroupIngress AWS SDK for JavaScriptAPIReference.

`CreateKeyPair`

Il seguente esempio di codice mostra come utilizzare`CreateKeyPair`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";
```

```
/**
 * Creates an ED25519 or 2048-bit RSA key pair with the specified name and in the
 specified PEM or PPK format.
 * Amazon EC2 stores the public key and displays the private key for you to save to
 a file.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new CreateKeyPairCommand({
    KeyName: keyName,
  });

  try {
    const { KeyMaterial, KeyName } = await client.send(command);
    console.log(KeyName);
    console.log(KeyMaterial);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidKeyPair.Duplicate") {
      console.warn(`${caught.message}. Try another key name.`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi CreateKeyPair AWS SDK for JavaScriptAPIReference.

`CreateLaunchTemplate`

Il seguente esempio di codice mostra come utilizzare`CreateLaunchTemplate`.

SDKper JavaScript (v3)

> **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.

```
  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 API i dettagli, vedi CreateLaunchTemplate AWS SDK for JavaScriptAPIReference.

`CreateSecurityGroup`

Il seguente esempio di codice mostra come utilizzare`CreateSecurityGroup`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates a security group.
 * @param {{ groupName: string, description: string }} options
```

```
 */
export const main = async ({ groupName, description }) => {
  const client = new EC2Client({});
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: groupName,
    // Up to 255 characters in length.
    Description: description,
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi CreateSecurityGroup AWS SDK for JavaScriptAPIReference.

`DeleteKeyPair`

Il seguente esempio di codice mostra come utilizzare`DeleteKeyPair`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes the specified key pair, by removing the public key from Amazon EC2.
```

```
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new DeleteKeyPairCommand({
    KeyName: keyName,
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide the required value?`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi DeleteKeyPair AWS SDK for JavaScriptAPIReference.

`DeleteLaunchTemplate`

Il seguente esempio di codice mostra come utilizzare`DeleteLaunchTemplate`.

SDKper JavaScript (v3)

> **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.

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- Per API i dettagli, vedi DeleteLaunchTemplate AWS SDK for JavaScriptAPIReference.

`DeleteSecurityGroup`

Il seguente esempio di codice mostra come utilizzare`DeleteSecurityGroup`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes a security group.
 * @param {{ groupId: string }} options
 */
export const main = async ({ groupId }) => {
  const client = new EC2Client({});
  const command = new DeleteSecurityGroupCommand({
    GroupId: groupId,
  });

  try {
    await client.send(command);
    console.log("Security group deleted successfully.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi DeleteSecurityGroup AWS SDK for JavaScriptAPIReference.

`DescribeAddresses`

Il seguente esempio di codice mostra come utilizzare`DescribeAddresses`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeAddressesCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Describes the specified Elastic IP addresses or all of your Elastic IP addresses.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
  const client = new EC2Client({});
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: [allocationId],
  });

  try {
    const { Addresses } = await client.send(command);
    const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
    console.log("Elastic IP addresses:");
    console.log(addressList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationId.`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi DescribeAddresses AWS SDK for JavaScriptAPIReference.

`DescribeIamInstanceProfileAssociations`

Il seguente esempio di codice mostra come utilizzare`DescribeIamInstanceProfileAssociations`.

SDKper JavaScript (v3)

> **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.

```
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
```

- Per API i dettagli, vedi DescribeIamInstanceProfileAssociations AWS SDK for JavaScriptAPIReference.

`DescribeImages`

Il seguente esempio di codice mostra come utilizzare`DescribeImages`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, paginateDescribeImages } from "@aws-sdk/client-ec2";

/**
 * Describes the specified images (AMIs, AKIs, and ARIs) available to you or all of
 the images available to you.
 * @param {{ architecture: string, pageSize: number }} options
 */
export const main = async ({ architecture, pageSize }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});

  // 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 },
    {
      // 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: [architecture] }],
    },
  );

  /**
   * @type {import('@aws-sdk/client-ec2').Image[]}
   */
  const images = [];
  let recordsScanned = 0;

  try {
    for await (const page of paginator) {
      recordsScanned += pageSize;
      if (page.Images.length) {
        images.push(...page.Images);
        break;
      } else {
        console.log(
          `No matching image found yet. Searched ${recordsScanned} records.`,
        );
      }
```

```
    }

    if (images.length) {
      console.log(
        `Found ${images.length} images:\n\n${images.map((image) =>
 image.Name).join("\n")}\n`,
      );
    } else {
      console.log(
        `No matching images found. Searched ${recordsScanned} records.\n`,
      );
    }

    return images;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
      return [];
    }
    throw caught;
  }
};
```

- Per API i dettagli, vedi DescribeImages AWS SDK for JavaScriptAPIReference.

`DescribeInstanceTypes`

Il seguente esempio di codice mostra come utilizzare`DescribeInstanceTypes`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, paginateDescribeInstanceTypes } from "@aws-sdk/client-ec2";

/**
 * Describes the specified instance types. By default, all instance types for the
```

```
 * current Region are described. Alternatively, you can filter the results.
 * @param {{ pageSize: string, supportedArch: string[], freeTier: boolean }} options
 */
export const main = async ({ pageSize, supportedArch, freeTier }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});

  // 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 },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          Values: supportedArch,
        },
        { Name: "free-tier-eligible", Values: [freeTier ? "true" : "false"] },
      ],
    },
  );

  try {
    /**
     * @type {import('@aws-sdk/client-ec2').InstanceTypeInfo[]}
     */
    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(
      `Memory size in MiB for matching instance types:\n\n${instanceTypes.map((it)
 => `${it.InstanceType}: ${it.MemoryInfo.SizeInMiB} MiB`).join("\n")}`,
    );
```

```
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
      return [];
    }
    throw caught;
  }
};
```

- Per API i dettagli, vedi DescribeInstanceTypes AWS SDK for JavaScriptAPIReference.

`DescribeInstances`

Il seguente esempio di codice mostra come utilizzare`DescribeInstances`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, paginateDescribeInstances } from "@aws-sdk/client-ec2";

/**
 * List all of your EC2 instances running with the provided architecture that
 * were launched in the past month.
 * @param {{ pageSize: string, architectures: string[] }} options
 */
export const main = async ({ pageSize, architectures }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;

  const paginator = paginateDescribeInstances(
    {
```

```
      client,
      pageSize,
    },
    {
      Filters: [
        { Name: "architecture", Values: architectures },
        { Name: "instance-state-name", Values: ["running"] },
        {
          Name: "launch-time",
          Values: [launchTimePattern],
        },
      ],
    },
  );

  try {
    /**
     * @type {import('@aws-sdk/client-ec2').Instance[]}
     */
    const instanceList = [];
    for await (const page of paginator) {
      const { Reservations } = page;
      Reservations.forEach((r) => instanceList.push(...r.Instances));
    }
    console.log(
      `Running instances launched this month:\n\n${instanceList.map((instance) =>
 instance.InstanceId).join("\n")}`,
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi DescribeInstances AWS SDK for JavaScriptAPIReference.

`DescribeKeyPairs`

Il seguente esempio di codice mostra come utilizzare`DescribeKeyPairs`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeKeyPairsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * List all key pairs in the current AWS account.
 * @param {{ dryRun: boolean }}
 */
export const main = async ({ dryRun }) => {
  const client = new EC2Client({});
  const command = new DescribeKeyPairsCommand({ DryRun: dryRun });

  try {
    const { KeyPairs } = await client.send(command);
    const keyPairList = KeyPairs.map(
      (kp) => ` • ${kp.KeyPairId}: ${kp.KeyName}`,
    ).join("\n");
    console.log("The following key pairs were found in your account:");
    console.log(keyPairList);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
      console.log(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi DescribeKeyPairs AWS SDK for JavaScriptAPIReference.

`DescribeRegions`

Il seguente esempio di codice mostra come utilizzare`DescribeRegions`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeRegionsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * List all available AWS regions.
 * @param {{ regionNames: string[], includeOptInRegions: boolean }} options
 */
export const main = async ({ regionNames, includeOptInRegions }) => {
  const client = new EC2Client({});
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions is true, even the regions that require opt-in will be
 returned.
    AllRegions: includeOptInRegions,
    // You can omit the Filters property if you want to get all regions.
    Filters: regionNames?.length
      ? [
          {
            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: regionNames,
          },
        ]
      : undefined,
  });

  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
```

```
      console.log(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi DescribeRegions AWS SDK for JavaScriptAPIReference.

`DescribeSecurityGroups`

Il seguente esempio di codice mostra come utilizzare`DescribeSecurityGroups`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Describes the specified security groups or all of your security groups.
 * @param {{ groupIds: string[] }} options
 */
export const main = async ({ groupIds = [] }) => {
  const client = new EC2Client({});
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: groupIds,
  });

  try {
    const { SecurityGroups } = await client.send(command);
    const sgList = SecurityGroups.map(
      (sg) => `• ${sg.GroupName} (${sg.GroupId}): ${sg.Description}`,
    ).join("\n");
    if (sgList.length) {
      console.log(`Security groups:\n${sgList}`);
```

```
    } else {
      console.log("No security groups found.");
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else if (
      caught instanceof Error &&
      caught.name === "InvalidGroup.NotFound"
    ) {
      console.warn(caught.message);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi DescribeSecurityGroups AWS SDK for JavaScriptAPIReference.

`DescribeSubnets`

Il seguente esempio di codice mostra come utilizzare`DescribeSubnets`.

SDKper JavaScript (v3)

> **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.

```
    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 API i dettagli, vedi DescribeSubnets AWS SDK for JavaScriptAPIReference.

`DescribeVpcs`

Il seguente esempio di codice mostra come utilizzare`DescribeVpcs`.

SDKper JavaScript (v3)

> **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.

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```

- Per API i dettagli, vedi DescribeVpcs AWS SDK for JavaScriptAPIReference.

`DisassociateAddress`

Il seguente esempio di codice mostra come utilizzare`DisassociateAddress`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DisassociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";
```

```
/**
 * Disassociate an Elastic IP address from an instance.
 * @param {{ associationId: string }} options
 */
export const main = async ({ associationId }) => {
  const client = new EC2Client({});
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AssociationId: associationId,
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAssociationID.NotFound"
    ) {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi DisassociateAddress AWS SDK for JavaScriptAPIReference.

`MonitorInstances`

Il seguente esempio di codice mostra come utilizzare`MonitorInstances`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, MonitorInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * 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.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new MonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instancesBeingMonitored = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instancesBeingMonitored.join("\n"));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi MonitorInstances AWS SDK for JavaScriptAPIReference.

`RebootInstances`

Il seguente esempio di codice mostra come utilizzare`RebootInstances`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, RebootInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Requests a reboot of the specified instances. This operation is asynchronous;
 * it only queues a request to reboot the specified instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new RebootInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Please provide the InstanceId of a valid instance to
 reboot.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi RebootInstances AWS SDK for JavaScriptAPIReference.

`ReleaseAddress`

Il seguente esempio di codice mostra come utilizzare`ReleaseAddress`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { ReleaseAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Release an Elastic IP address.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
  const client = new EC2Client({});
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AllocationId: allocationId,
  });

  try {
    await client.send(command);
    console.log("Successfully released address.");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationID.`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi ReleaseAddress AWS SDK for JavaScriptAPIReference.

`ReplaceIamInstanceProfileAssociation`

Il seguente esempio di codice mostra come utilizzare`ReplaceIamInstanceProfileAssociation`.

SDKper JavaScript (v3)

> **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.

```
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
);
```

- Per API i dettagli, vedi ReplaceIamInstanceProfileAssociation AWS SDK for JavaScriptAPIReference.

`RunInstances`

Il seguente esempio di codice mostra come utilizzare`RunInstances`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, RunInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Create new EC2 instances.
 * @param {{
 *  keyName: string,
 *  securityGroupIds: string[],
 *  imageId: string,
 *  instanceType: import('@aws-sdk/client-ec2')._InstanceType,
 *  minCount?: number,
 *  maxCount?: number }} options
 */
export const main = async ({
  keyName,
  securityGroupIds,
  imageId,
  instanceType,
  minCount = "1",
  maxCount = "1",
}) => {
  const client = new EC2Client({});
  minCount = parseInt(minCount);
  maxCount = parseInt(maxCount);
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: keyName,
    // Your security group.
    SecurityGroupIds: securityGroupIds,
    // An Amazon Machine Image (AMI). There are multiple ways to search for AMIs.
 For more information, see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html
    ImageId: imageId,
    // An instance type describing the resources provided to your instance. There
 are multiple
    // ways to search for instance types. For more information see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-discovery.html
    InstanceType: instanceType,
    // Availability Zones have capacity limitations that may impact your ability to
 launch instances.
    // The `RunInstances` operation will only succeed if it can allocate at least
 the `MinCount` of instances.
    // However, EC2 will attempt to launch up to the `MaxCount` of instances, even
 if the full request cannot be satisfied.
```

```
    // If you need a specific number of instances, use `MinCount` and `MaxCount` set
 to the same value.
    // If you want to launch up to a certain number of instances, use `MaxCount` and
 let EC2 provision as many as possible.
    // If you require a minimum number of instances, but do not want to exceed a
 maximum, use both `MinCount` and `MaxCount`.
    MinCount: minCount,
    MaxCount: maxCount,
  });

  try {
    const { Instances } = await client.send(command);
    const instanceList = Instances.map(
      (instance) => `• ${instance.InstanceId}`,
    ).join("\n");
    console.log(`Launched instances:\n${instanceList}`);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceCountExceeded") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi RunInstances AWS SDK for JavaScriptAPIReference.

`StartInstances`

Il seguente esempio di codice mostra come utilizzare`StartInstances`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, StartInstancesCommand } from "@aws-sdk/client-ec2";
```

```
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Starts an Amazon EBS-backed instance that you've previously stopped.
 * @param {{ instanceIds }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StartInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi StartInstances AWS SDK for JavaScriptAPIReference.

`StopInstances`

Il seguente esempio di codice mostra come utilizzare`StopInstances`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, StopInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Stop one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StopInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi StopInstances AWS SDK for JavaScriptAPIReference.

`TerminateInstances`

Il seguente esempio di codice mostra come utilizzare`TerminateInstances`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, TerminateInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Terminate one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new TerminateInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
```

```
      throw caught;
    }
  }
  ``;
};
```

- Per API i dettagli, vedi TerminateInstances AWS SDK for JavaScriptAPIReference.

`UnmonitorInstances`

Il seguente esempio di codice mostra come utilizzare`UnmonitorInstances`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { EC2Client, UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Turn off detailed monitoring for the selected instance.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new UnmonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  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 (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi UnmonitorInstances AWS SDK for JavaScriptAPIReference.

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 Amazon EC2 Auto Scaling per creare istanze Amazon Elastic Compute Cloud (AmazonEC2) basate su un modello di avvio e per mantenere il numero di istanze in un intervallo specificato.
- Gestisci e distribuisci HTTP le richieste 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 EC2 istanza 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 di integrità aggiornando AWS Systems Manager i parametri.

SDKper JavaScript (v3)

> **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.

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.

```
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 { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 () => {
    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,
        },
      }),
    );
  }),
  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) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    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) => {
    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"] },
        ],
      }),
    );
```

```
    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) => {
    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,
      }),
    );
    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("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${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) => {
    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 },
        ],
      }),
    );
    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) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  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,
      );
    }
  }),
  saveState,
];
```

Crea i passaggi per eseguire la demo.

```
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) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
```

```
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
```

```
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
```

```
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
```

```
        }),
      );
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );

      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",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Crea i passaggi per distruggere tutte le risorse.

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} 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 { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 = [
  loadState,
  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 {
      const client = new IAMClient({});
      await client.send(
```

```
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } 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 {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } 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 {
      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.");
        }
      });
    } 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) => {
    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;
    }
  }),
  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,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
```

```
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - 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

Elastic Load Balancing - Esempi della versione 2 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 - Version 2.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui 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.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 API i dettagli, vedi DescribeLoadBalancers AWS SDK for JavaScriptAPIReference.

Argomenti

- Azioni
- Scenari

Azioni

`CreateListener`

Il seguente esempio di codice mostra come utilizzare`CreateListener`.

SDKper JavaScript (v3)

> **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.

```
    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 API i dettagli, vedi CreateListener AWS SDK for JavaScriptAPIReference.

`CreateLoadBalancer`

Il seguente esempio di codice mostra come utilizzare`CreateLoadBalancer`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi CreateLoadBalancer AWS SDK for JavaScriptAPIReference.

`CreateTargetGroup`

Il seguente esempio di codice mostra come utilizzare`CreateTargetGroup`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi CreateTargetGroup AWS SDK for JavaScriptAPIReference.

`DeleteLoadBalancer`

Il seguente esempio di codice mostra come utilizzare`DeleteLoadBalancer`.

SDKper JavaScript (v3)

> **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.

```
  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 API i dettagli, vedi DeleteLoadBalancer AWS SDK for JavaScriptAPIReference.

`DeleteTargetGroup`

Il seguente esempio di codice mostra come utilizzare`DeleteTargetGroup`.

SDKper JavaScript (v3)

> **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.

```
  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 API i dettagli, vedi DeleteTargetGroup AWS SDK for JavaScriptAPIReference.

`DescribeLoadBalancers`

Il seguente esempio di codice mostra come utilizzare`DescribeLoadBalancers`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 API i dettagli, vedi DescribeLoadBalancers AWS SDK for JavaScriptAPIReference.

`DescribeTargetGroups`

Il seguente esempio di codice mostra come utilizzare`DescribeTargetGroups`.

SDKper JavaScript (v3)

> **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.

```
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
```

- Per API i dettagli, vedi DescribeTargetGroups AWS SDK for JavaScriptAPIReference.

`DescribeTargetHealth`

Il seguente esempio di codice mostra come utilizzare`DescribeTargetHealth`.

SDKper JavaScript (v3)

> **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.

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- Per API i dettagli, vedi DescribeTargetHealth AWS SDK for JavaScriptAPIReference.

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 Amazon EC2 Auto Scaling per creare istanze Amazon Elastic Compute Cloud (AmazonEC2) basate su un modello di avvio e per mantenere il numero di istanze in un intervallo specificato.
- Gestisci e distribuisci HTTP le richieste 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 EC2 istanza 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 di integrità aggiornando AWS Systems Manager i parametri.

SDKper JavaScript (v3)

> **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.

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.

```
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 { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 () => {
    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,
        },
      }),
    );
  }),
  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) => {
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  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) => {
  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"] },
      ],
    }),
  );
  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) => {
  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,
    }),
  );
  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) => {
  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] },
  );
}),
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) => {
  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 },
      ],
```

```
      }),
    );
    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) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  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,
      );
    }
  }),
  saveState,
];
```

Crea i passaggi per eseguire la demo.

```
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) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
```

```
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
```

```
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
```

```
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
```

```
    new DescribeIamInstanceProfileAssociationsCommand({
      Filters: [
        { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
      ],
    }),
  );
  state.instanceProfileAssociationId =
    IamInstanceProfileAssociations[0].AssociationId;
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    ec2Client.send(
      new ReplaceIamInstanceProfileAssociationCommand({
        AssociationId: state.instanceProfileAssociationId,
        IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
      }),
    ),
  );

  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",
  }),
);
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
```

```
    }),
  );

  return InstanceProfile;
}
```

Crea i passaggi per distruggere tutte le risorse.

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} 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 { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 = [
  loadState,
  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 {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } 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(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      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.");
      }
    });
  } 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) => {
  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;
  }
}),
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,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
```

```
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - 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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni
- Scenari

Azioni

`PutEvents`

Il seguente esempio di codice mostra come utilizzare`PutEvents`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere PutEventsin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutEvents AWS SDK for JavaScriptAPIReference.

`PutRule`

Il seguente esempio di codice mostra come utilizzare`PutRule`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere PutRulein AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutRule AWS SDK for JavaScriptAPIReference.

`PutTargets`

Il seguente esempio di codice mostra come utilizzare`PutTargets`.

SDKper JavaScript (v3)

> **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.

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere PutTargetsin AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi PutTargets AWS SDK for JavaScriptAPIReference.

Scenari

Utilizzo degli eventi pianificati per richiamare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

SDKper 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, si crea una funzione Lambda utilizzando il runtime Lambda. JavaScript API 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

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

Gli elementi di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto nei relativi scenari.

Ogni esempio include un collegamento al codice sorgente completo, in cui 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.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListJobs AWS SDK for JavaScriptAPIReference.

Argomenti

- Nozioni di base
- Azioni

Nozioni di base

Scopri le nozioni di base

L'esempio di codice seguente mostra come:

- Crea un crawler che esegua la scansione di un bucket Amazon S3 pubblico e generi un database di metadati formattati. CSV
- Elenca le informazioni su database e tabelle nel tuo. AWS Glue Data Catalog
- Crea un job per estrarre CSV dati dal bucket S3, trasformarli e caricare l'output in JSON formato S3 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 Studio. AWS Glue

SDKper JavaScript (v3)

> **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.

Crea ed esegui un crawler che esegua la scansione di un bucket Amazon Simple Storage Service (Amazon S3) pubblico e generi un database di metadati che descrive i dati in formato che trova. 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;
  }
};

/**
```

```
 * @param {{ createCrawler: import('../../../actions/create-
crawler.js').createCrawler}} actions
 */
const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH,
    );

    log("Crawler created successfully.", { type: "success" });
  }

  return { ...context };
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-
glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }

  log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
  await wait(waitTimeInSeconds);
  return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
```

```
  ({ startCrawler, getCrawler }) =>
  async (context) => {
    log("Starting crawler.");
    await startCrawler(process.env.CRAWLER_NAME);
    log("Crawler started.", { type: "success" });

    log("Waiting for crawler to finish running. This can take a while.");
    await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
    log("Crawler ready.", { type: "success" });

    return { ...context };
  };
```

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 };
  };

/**
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-
glue').GetTablesCommandOutput}} config
 */
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
```

Crea ed esegui un processo che estrae CSV i dati dal bucket Amazon S3 di origine, li trasforma rimuovendo e rinominando i campi e JSON carica l'output formattato 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);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput>}
 getJobRun
 */
```

```
/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunsCommandOutput}
 getJobRuns
 */

/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
 *
 * @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
 */
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (/** @type { Context } */ context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };
```

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);
};

/**
 *
 * @param {import('../../../actions/delete-job.js').deleteJob} deleteJobFn
 * @param {string[]} jobNames
```

```
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
   * @type {{ selectedJobNames: string[] }}
   */
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error)),
    );
    log("Jobs deleted.", { type: "success" });
  }
};

/**
 * @param {{
 *   listJobs: import('../../../actions/list-jobs.js').listJobs,
 *   deleteJob: import('../../../actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

/**
 * @param {import('../../../actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
```

```
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

/**
 * @param {{
 *  getTables: import('../../../actions/get-tables.js').getTables,
 *  deleteTable: import('../../../actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}}} context
   */
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null }),
    );

    if (TableList && TableList.length > 0) {
      /**
       * @type {{ tableNames: string[] }}
       */
      const { tableNames } = await context.prompter.prompt({
        name: "tableNames",
        type: "checkbox",
        message: "Let's clean up tables. Select tables to delete.",
        choices: TableList.map((t) => t.Name),
      });

      if (tableNames.length === 0) {
        log("No tables selected.");
      } else {
        log("Deleting tables.");
        await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
        log("Tables deleted.", { type: "success" });
      }
    }
```

```
    return { ...context };
  };

/**
 * @param {import('../../../actions/delete-database.js').deleteDatabase}
 deleteDatabase
 * @param {string[]} databaseNames
 */
const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('../../../actions/get-databases.js').getDatabases
 *   deleteDatabase: import('../../../actions/delete-database.js').deleteDatabase
 * }} config
 */
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}} context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbNames: string[] }} */
      const { dbNames } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbNames.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
      }
```

```
    }

    return { ...context };
  };

const cleanUpCrawlerStep = async (context) => {
  log(`Deleting crawler.`);

  try {
    await deleteCrawler(process.env.CRAWLER_NAME);
    log("Crawler deleted.", { type: "success" });
  } catch (err) {
    if (err.name === "EntityNotFoundException") {
      log(`Crawler is already deleted.`);
    } else {
      throw err;
    }
  }

  return { ...context };
};
```

- Per API i dettagli, consulta i seguenti argomenti in Reference.AWS SDK for JavaScript API
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables

- ListJobs
- StartCrawler
- StartJobRun

Azioni

`CreateCrawler`

Il seguente esempio di codice mostra come utilizzare`CreateCrawler`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi CreateCrawler AWS SDK for JavaScriptAPIReference.

`CreateJob`

Il seguente esempio di codice mostra come utilizzare`CreateJob`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi CreateJob AWS SDK for JavaScriptAPIReference.

`DeleteCrawler`

Il seguente esempio di codice mostra come utilizzare`DeleteCrawler`.

SDKper JavaScript (v3)

> **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.

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi DeleteCrawler AWS SDK for JavaScriptAPIReference.

`DeleteDatabase`

Il seguente esempio di codice mostra come utilizzare`DeleteDatabase`.

SDKper JavaScript (v3)

> **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.

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });
```

```
  return client.send(command);
};
```

- Per API i dettagli, vedi DeleteDatabase AWS SDK for JavaScriptAPIReference.

`DeleteJob`

Il seguente esempio di codice mostra come utilizzare`DeleteJob`.

SDKper JavaScript (v3)

> **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.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi DeleteJob AWS SDK for JavaScriptAPIReference.

`DeleteTable`

Il seguente esempio di codice mostra come utilizzare`DeleteTable`.

SDKper JavaScript (v3)

> **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.

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi DeleteTable AWS SDK for JavaScriptAPIReference.

`GetCrawler`

Il seguente esempio di codice mostra come utilizzare`GetCrawler`.

SDKper JavaScript (v3)

> **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.

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
```

```
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi GetCrawler AWS SDK for JavaScriptAPIReference.

`GetDatabase`

Il seguente esempio di codice mostra come utilizzare`GetDatabase`.

SDKper JavaScript (v3)

> **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.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi GetDatabase AWS SDK for JavaScriptAPIReference.

`GetDatabases`

Il seguente esempio di codice mostra come utilizzare`GetDatabases`.

SDKper JavaScript (v3)

> **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.

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- Per API i dettagli, vedi GetDatabases AWS SDK for JavaScriptAPIReference.

`GetJob`

Il seguente esempio di codice mostra come utilizzare`GetJob`.

SDKper JavaScript (v3)

> **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.

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi GetJob AWS SDK for JavaScriptAPIReference.

`GetJobRun`

Il seguente esempio di codice mostra come utilizzare`GetJobRun`.

SDKper JavaScript (v3)

> **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.

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi GetJobRun AWS SDK for JavaScriptAPIReference.

`GetJobRuns`

Il seguente esempio di codice mostra come utilizzare`GetJobRuns`.

SDKper JavaScript (v3)

> **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.

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi GetJobRuns AWS SDK for JavaScriptAPIReference.

`GetTables`

Il seguente esempio di codice mostra come utilizzare`GetTables`.

SDKper JavaScript (v3)

> **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.

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- Per API i dettagli, vedi GetTables AWS SDK for JavaScriptAPIReference.

`ListJobs`

Il seguente esempio di codice mostra come utilizzare`ListJobs`.

SDKper JavaScript (v3)

> **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.

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- Per API i dettagli, vedi ListJobs AWS SDK for JavaScriptAPIReference.

`StartCrawler`

Il seguente esempio di codice mostra come utilizzare`StartCrawler`.

SDKper JavaScript (v3)

> **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.

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
```

```
};
```

- Per API i dettagli, vedi StartCrawler AWS SDK for JavaScriptAPIReference.

`StartJobRun`

Il seguente esempio di codice mostra come utilizzare`StartJobRun`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi StartJobRun AWS SDK for JavaScriptAPIReference.

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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve HealthImaging

I seguenti esempi di codice mostrano come iniziare a utilizzare HealthImaging.

SDKper 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 API i dettagli, vedere ListDatastoresin AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Argomenti

Azioni

`CopyImageSet`

Il seguente esempio di codice mostra come usare`CopyImageSet`.

SDKper 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.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
    datastoreId = "xxxxxxxxxx",
    imageSetId = "xxxxxxxxxxx",
    sourceVersionId = "1",
    destinationImageSetId = "",
    destinationVersionId = "",
    force = false,
```

```
    copySubsets = []
) => {
    try {
        const params = {
            datastoreId: datastoreId,
            sourceImageSetId: imageSetId,
            copyImageSetInformation: {
                sourceImageSet: {latestVersionId: sourceVersionId},
            },
            force: force
        };
        if (destinationImageSetId !== "" && destinationVersionId !== "") {
            params.copyImageSetInformation.destinationImageSet = {
                imageSetId: destinationImageSetId,
                latestVersionId: destinationVersionId,
            };
        }

        if (copySubsets.length > 0) {
            let copySubsetsJson;
            copySubsetsJson = {
                SchemaVersion: 1.1,
                Study: {
                    Series: {
                        imageSetId: {
                            Instances: {}
                        }
                    }
                }
            };

            for (let i = 0; i < copySubsets.length; i++) {
                copySubsetsJson.Study.Series.imageSetId.Instances[
                    copySubsets[i]
                    ] = {};
            }

            params.copyImageSetInformation.dicomCopies = copySubsetsJson;
        }

        const response = await medicalImagingClient.send(
            new CopyImageSetCommand(params)
        );
        console.log(response);
```

```
        // {
        //     '$metadata': {
        //         httpStatusCode: 200,
        //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
        //         extendedRequestId: undefined,
        //         cfId: undefined,
        //         attempts: 1,
        //         totalRetryDelay: 0
        //      },
        //       datastoreId: 'xxxxxxxxxxxxx',
        //       destinationImageSetProperties: {
        //             createdAt: 2023-09-27T19:46:21.824Z,
        //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
        //             imageSetId: 'xxxxxxxxxxxxxx',
        //             imageSetState: 'LOCKED',
        //             imageSetWorkflowStatus: 'COPYING',
        //             latestVersionId: '1',
        //             updatedAt: 2023-09-27T19:46:21.824Z
        //       },
        //       sourceImageSetProperties: {
        //             createdAt: 2023-09-22T14:49:26.427Z,
        //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
        //             imageSetId: 'xxxxxxxxxxxxxx',
        //             imageSetState: 'LOCKED',
        //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
        //             latestVersionId: '4',
        //             updatedAt: 2023-09-27T19:46:21.824Z
        //     }
        // }
        return response;
    } catch (err) {
        console.error(err);
    }
};
```

Copia un set di immagini senza una destinazione.

```
    await copyImageSet(
        "12345678901234567890123456789012",
```

```
        "12345678901234567890123456789012",
        "1"
    );
```

Copia un set di immagini con una destinazione.

```
    await copyImageSet(
        "12345678901234567890123456789012",
        "12345678901234567890123456789012",
        "1",
        "12345678901234567890123456789012",
        "1",
        false,
    );
```

Copia un sottoinsieme di un set di immagini con una destinazione e forza la copia.

```
    await copyImageSet(
        "12345678901234567890123456789012",
        "12345678901234567890123456789012",
        "1",
        "12345678901234567890123456789012",
        "1",
        true,
        ["12345678901234567890123456789012", "11223344556677889900112233445566"]
    );
```

- Per API i dettagli, vedere CopyImageSetin AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`CreateDatastore`

Il seguente esempio di codice mostra come usare`CreateDatastore`.

SDKper 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 API i dettagli, vedere CreateDatastorein AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`DeleteDatastore`

Il seguente esempio di codice mostra come usare`DeleteDatastore`.

SDKper 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 API i dettagli, vedere DeleteDatastorein AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`DeleteImageSet`

Il seguente esempio di codice mostra come usare`DeleteImageSet`.

SDKper JavaScript (v3)

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //    '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Per API i dettagli, vedere DeleteImageSetin AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`GetDICOMImportJob`

Il seguente esempio di codice mostra come usare`GetDICOMImportJob`.

SDKper 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 API i dettagli, vedere G etDICOMImport Job in AWS SDK for JavaScript APIReference.

> **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.

`GetDatastore`

Il seguente esempio di codice mostra come usare`GetDatastore`.

SDKper 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 API i dettagli, vedere GetDatastorein AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`GetImageFrame`

Il seguente esempio di codice mostra come usare`GetImageFrame`.

SDKper 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 API i dettagli, vedere GetImageFramein AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`GetImageSet`

Il seguente esempio di codice mostra come usare`GetImageSet`.

SDKper 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: 'xxxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }
```

```
  return response;
};
```

- Per API i dettagli, vedere GetImageSetin AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`GetImageSetMetadata`

Il seguente esempio di codice mostra come usare`GetImageSetMetadata`.

SDKper 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 = "xxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxx",
  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 API i dettagli, consulta la sezione GetImageSetMetadata AWS SDK for JavaScriptAPIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`ListDICOMImportJobs`

Il seguente esempio di codice mostra come usare`ListDICOMImportJobs`.

SDKper 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 API i dettagli, vedere L istDICOMImport Jobs in AWS SDK for JavaScript APIReference.

> **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.

`ListDatastores`

Il seguente esempio di codice mostra come usare`ListDatastores`.

SDKper 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 API i dettagli, vedere ListDatastoresin AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`ListImageSetVersions`

Il seguente esempio di codice mostra come usare`ListImageSetVersions`.

SDKper 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 = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );
```

```
  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Per API i dettagli, vedere ListImageSetVersionsin AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`ListTagsForResource`

Il seguente esempio di codice mostra come usare`ListTagsForResource`.

SDKper 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 API i dettagli, vedere ListTagsForResourcein AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`SearchImageSets`

Il seguente esempio di codice mostra come usare`SearchImageSets`.

SDKper 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.
 * @param { import('@aws-sdk/client-medical-imaging').Sort } sort - The search
 criteria sort.
 */
export const searchImageSets = async (
    datastoreId = "xxxxxxxx",
    searchCriteria = {}
) => {
    const paginatorConfig = {
        client: medicalImagingClient,
        pageSize: 50,
    };

    const commandParams = {
        datastoreId: datastoreId,
        searchCriteria: searchCriteria,
    };

    const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

    const imageSetsMetadataSummaries = [];
    for await (const page of paginator) {
        // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
        imageSetsMetadataSummaries.push(...page["imageSetsMetadataSummaries"]);
        console.log(page);
    }
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
```

```
    //         requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //     },
    //     imageSetsMetadataSummaries: [
    //         {
    //             DICOMTags: [Object],
    //             createdAt: "2023-09-19T16:59:40.551Z",
    //             imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',
    //             updatedAt: "2023-09-19T16:59:40.551Z",
    //             version: 1
    //         }]
    // }

    return imageSetsMetadataSummaries;
};
```

Caso d'uso #1: EQUAL operatore.

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [{DICOMPatientId: "1234567"}],
                    operator: "EQUAL",
                },
            ]
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

Caso d'uso #2: BETWEEN operatore che utilizza DICOMStudyDate andDICOMStudyTime.

```
    const datastoreId = "12345678901234567890123456789012";
```

```
try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {
                        DICOMStudyDateAndTime: {
                            DICOMStudyDate: "19900101",
                            DICOMStudyTime: "000000",
                        },
                    },
                    {
                        DICOMStudyDateAndTime: {
                            DICOMStudyDate: "20230901",
                            DICOMStudyTime: "000000",
                        },
                    },
                ],
                operator: "BETWEEN",
            },
        ]
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

Caso d'uso #3: BETWEEN operatore che utilizzacreatedAt. Gli studi sul tempo erano stati precedentemente proseguiti.

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {createdAt: new Date("1985-04-12T23:20:50.52Z")},
                    {createdAt: new Date()},
                ],
```

```
                    operator: "BETWEEN",
                },
            ]
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
    }
```

Caso d'uso #4: EQUAL operatore acceso DICOMSeriesInstanceUID updatedAt e BETWEEN ordina la risposta in ASC ordine sul updatedAt campo.

```
    const datastoreId = "12345678901234567890123456789012";

    try {
        const searchCriteria = {
            filters: [
                {
                    values: [
                        {updatedAt: new Date("1985-04-12T23:20:50.52Z")},
                        {updatedAt: new Date()},
                    ],
                    operator: "BETWEEN",
                },
                {
                    values: [
                        {DICOMSeriesInstanceUID:
"1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
                    ],
                    operator: "EQUAL",
                },
            ],
            sort: {
                sortOrder: "ASC",
                sortField: "updatedAt",
            }
        };

        await searchImageSets(datastoreId, searchCriteria);
    } catch (err) {
        console.error(err);
```

```
    }
```

- Per API i dettagli, vedere SearchImageSetsin AWS SDK for JavaScript APIRiferimento.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`StartDICOMImportJob`

Il seguente esempio di codice mostra come usare`StartDICOMImportJob`.

SDKper 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 API i dettagli, vedere S tartDICOMImport Job in AWS SDK for JavaScript APIReference.

> **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.

`TagResource`

Il seguente esempio di codice mostra come usare`TagResource`.

SDKper 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 API i dettagli, vedere TagResourcein AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`UntagResource`

Il seguente esempio di codice mostra come usare`UntagResource`.

SDKper 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 API i dettagli, vedere UntagResourcein AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

`UpdateImageSetMetadata`

Il seguente esempio di codice mostra come usare`UpdateImageSetMetadata`.

SDKper 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.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}',
                                             force = false) => {
    try {
        const response = await medicalImagingClient.send(
            new UpdateImageSetMetadataCommand({
                datastoreId: datastoreId,
                imageSetId: imageSetId,
                latestVersionId: latestVersionId,
                updateImageSetMetadataUpdates: updateMetadata,
                force: force,
            })
        );
        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;
    } catch (err) {
```

```
        console.error(err);
    }
};
```

Caso d'uso #1: inserire o aggiornare un attributo e forzare l'aggiornamento.

```
        const insertAttributes =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "updatableAttributes":
                    new TextEncoder().encode(insertAttributes)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata, true);
```

Caso d'uso #2: rimuovere un attributo.

```
        // Attribute key and value must match the existing attribute.
        const remove_attribute =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "DICOM": {
                        "StudyDescription": "CT CHEST"
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
```

```
                "removableAttributes":
                    new TextEncoder().encode(remove_attribute)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

Caso d'uso #3: rimuove un'istanza.

```
        const remove_instance =
            JSON.stringify({
                "SchemaVersion": 1.1,
                "Study": {
                    "Series": {
                        "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
                            "Instances": {

"1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
                            }
                        }
                    }
                }
            });

        const updateMetadata = {
            "DICOMUpdates": {
                "removableAttributes":
                    new TextEncoder().encode(remove_instance)
            }
        };

        await updateImageSetMetadata(datastoreID, imageSetID,
            versionID, updateMetadata);
```

Caso d'uso #4: ripristina una versione precedente.

```
    const updateMetadata = {
        "revertToVersionId": "1"
    };
```

```
    await updateImageSetMetadata(datastoreID, imageSetID,
        versionID, updateMetadata);
```

- Per API i dettagli, vedere UpdateImageSetMetadatain AWS SDK for JavaScript APIReference.

> **Note**
>
> C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Scenari

Inizia con set di immagini e cornici di immagini

Il seguente esempio di codice mostra come importare DICOM file e scaricare cornici di immagini in HealthImaging.

L'implementazione è strutturata come un'applicazione a riga di comando per il flusso di lavoro.

- Imposta le risorse per un'DICOMimportazione.
- Importa DICOM file in un archivio dati.
- Recuperate il set di immagini IDs per il processo di importazione.
- Recuperate la cornice dell'immagine IDs per i set di immagini.
- Scarica, decodifica e verifica le cornici dell'immagine.
- Pulisci le risorse.

SDKper JavaScript (v3)

index.js- Orchestra i passaggi.

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
```

```
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
```

```
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
```

```
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

deploy-steps.js- Distribuisci risorse.

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
```

```
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });
```

```
    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName == state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
```

```
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

dataset-steps.js- Copiare DICOM i file.

```
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
```

```
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;
```

```
    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js- Inizia l'importazione nel datastore.

```
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
```

```
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);
```

```
export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

image-set-steps.js- Ottieni un set di immagini. IDs

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

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
```

```
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);
```

```
export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js- Ottieni una cornice per l'immagineIDs.

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */
```

```
/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
```

```
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
```

```
    }

    return output;
  },
  { slow: false },
);
```

verify-steps.js- Verifica le cornici delle immagini. La libreria AWS HealthImaging Pixel Data Verification è stata utilizzata per la verifica.

```
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
```

```
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
    default: true,
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
```

```
 async (/** @type {State} */ state) => {
   if (!state.doVerify) {
     process.exit(0);
   }
   const verificationTool = "./pixel-data-verification/index.js";

   for (const metadata of state.imageSetMetadata) {
     const datastoreId = state.stackOutputs.DatastoreID;
     const imageSetId = metadata.ImageSetID;

     for (const [seriesInstanceUid, series] of Object.entries(
       metadata.Study.Series,
     )) {
       for (const [sopInstanceUid, _] of Object.entries(series.Instances)) {
         console.log(
           `Verifying image set ${imageSetId} with series ${seriesInstanceUid} and
sop ${sopInstanceUid}`,
         );
         const child = spawn(
           "node",
           [
             verificationTool,
             datastoreId,
             imageSetId,
             seriesInstanceUid,
             sopInstanceUid,
           ],
           { stdio: "inherit" },
         );

         await new Promise((resolve, reject) => {
           child.on("exit", (code) => {
             if (code === 0) {
               resolve();
             } else {
               reject(
                 new Error(
                   `Verification tool exited with code ${code} for image set
${imageSetId}`,
                 ),
               );
             }
           });
         });
```

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

clean-up-steps.js- Distruggi le risorse.

```
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
```

```
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
```

```
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
```

```
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- Per API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIReference.
  - DeleteImageSet
  - G etDICOMImport Job
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - S tartDICOMImport Job

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Etichettare un archivio dati

Il seguente esempio di codice mostra come etichettare un HealthImaging data store.

SDKper 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 API i dettagli, consultate i seguenti argomenti in AWS SDK for JavaScript API Reference.
  - ListTagsForResource

- TagResource
- UntagResource

> **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.

Etichettare un set di immagini

Il seguente esempio di codice mostra come etichettare un set di HealthImaging immagini.

SDKper 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 API i dettagli, consultate i seguenti argomenti in AWS SDK for JavaScript API Reference.
  - ListTagsForResource
  - TagResource
  - UntagResource

> **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.

IAMesempi che usano 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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello IAM

I seguenti esempi di codice mostrano come iniziare a utilizzareIAM.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListPolicies AWS SDK for JavaScriptAPIReference.

Argomenti

- Azioni
- Scenari

Azioni

`AttachRolePolicy`

Il seguente esempio di codice mostra come utilizzare`AttachRolePolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi AttachRolePolicy AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi AttachRolePolicy AWS SDK for JavaScriptAPIReference.

`CreateAccessKey`

Il seguente esempio di codice mostra come utilizzare`CreateAccessKey`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi CreateAccessKey AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi CreateAccessKey AWS SDK for JavaScriptAPIReference.

`CreateAccountAlias`

Il seguente esempio di codice mostra come utilizzare`CreateAccountAlias`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi CreateAccountAlias AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi CreateAccountAlias AWS SDK for JavaScriptAPIReference.

`CreateGroup`

Il seguente esempio di codice mostra come utilizzare`CreateGroup`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi CreateGroup AWS SDK for JavaScriptAPIReference.

`CreateInstanceProfile`

Il seguente esempio di codice mostra come utilizzare`CreateInstanceProfile`.

SDKper JavaScript (v3)

> **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.

```
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```

- Per API i dettagli, vedi CreateInstanceProfile AWS SDK for JavaScriptAPIReference.

`CreatePolicy`

Il seguente esempio di codice mostra come utilizzare`CreatePolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi CreatePolicy AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi CreatePolicy AWS SDK for JavaScriptAPIReference.

`CreateRole`

Il seguente esempio di codice mostra come utilizzare`CreateRole`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi CreateRole AWS SDK for JavaScriptAPIReference.

`CreateSAMLProvider`

Il seguente esempio di codice mostra come utilizzare`CreateSAMLProvider`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedere C reateSAMLProvider in AWS SDK for JavaScript APIReference.

`CreateServiceLinkedRole`

Il seguente esempio di codice mostra come utilizzare`CreateServiceLinkedRole`.

SDKper JavaScript (v3)

> **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.

Crea un ruolo collegato ai servizi.

```
import {
  CreateServiceLinkedRoleCommand,
  GetRoleCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-
that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/
latest/gr/aws-service-information.html.
```

```
    AWSServiceName: serviceName,
  });
  try {
    const response = await client.send(command);
    console.log(response);
    return response;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInputException" &&
      caught.message.includes(
        "Service role name AWSServiceRoleForElasticBeanstalk has been taken in this
 account",
      )
    ) {
      console.warn(caught.message);
      return client.send(
        new GetRoleCommand({ RoleName: "AWSServiceRoleForElasticBeanstalk" }),
      );
    } else {
      throw caught;
    }
  }
};
```

- Per API i dettagli, vedi CreateServiceLinkedRole AWS SDK for JavaScriptAPIReference.

`CreateUser`

Il seguente esempio di codice mostra come utilizzare`CreateUser`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi CreateUser AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi CreateUser AWS SDK for JavaScriptAPIReference.

`DeleteAccessKey`

Il seguente esempio di codice mostra come utilizzare`DeleteAccessKey`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteAccessKey AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DeleteAccessKey AWS SDK for JavaScriptAPIReference.

`DeleteAccountAlias`

Il seguente esempio di codice mostra come utilizzare`DeleteAccountAlias`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteAccountAlias AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DeleteAccountAlias AWS SDK for JavaScriptAPIReference.

`DeleteGroup`

Il seguente esempio di codice mostra come utilizzare`DeleteGroup`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteGroup AWS SDK for JavaScriptAPIReference.

`DeleteInstanceProfile`

Il seguente esempio di codice mostra come utilizzare`DeleteInstanceProfile`.

SDKper JavaScript (v3)

> **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.

```
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
```

- Per API i dettagli, vedi DeleteInstanceProfile AWS SDK for JavaScriptAPIReference.

`DeletePolicy`

Il seguente esempio di codice mostra come utilizzare`DeletePolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeletePolicy AWS SDK for JavaScriptAPIReference.

`DeleteRole`

Il seguente esempio di codice mostra come utilizzare`DeleteRole`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteRole AWS SDK for JavaScriptAPIReference.

`DeleteRolePolicy`

Il seguente esempio di codice mostra come utilizzare`DeleteRolePolicy`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteRolePolicy AWS SDK for JavaScriptAPIReference.

`DeleteSAMLProvider`

Il seguente esempio di codice mostra come utilizzare`DeleteSAMLProvider`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedere D eleteSAMLProvider in AWS SDK for JavaScript APIReference.

`DeleteServerCertificate`

Il seguente esempio di codice mostra come utilizzare`DeleteServerCertificate`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteServerCertificate AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DeleteServerCertificate AWS SDK for JavaScriptAPIReference.

`DeleteServiceLinkedRole`

Il seguente esempio di codice mostra come utilizzare`DeleteServiceLinkedRole`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteServiceLinkedRole AWS SDK for JavaScriptAPIReference.

`DeleteUser`

Il seguente esempio di codice mostra come utilizzare`DeleteUser`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteUser AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DeleteUser AWS SDK for JavaScriptAPIReference.

`DetachRolePolicy`

Il seguente esempio di codice mostra come utilizzare`DetachRolePolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DetachRolePolicy AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi DetachRolePolicy AWS SDK for JavaScriptAPIReference.

`GetAccessKeyLastUsed`

Il seguente esempio di codice mostra come utilizzare`GetAccessKeyLastUsed`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi GetAccessKeyLastUsed AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi GetAccessKeyLastUsed AWS SDK for JavaScriptAPIReference.

`GetAccountPasswordPolicy`

Il seguente esempio di codice mostra come utilizzare`GetAccountPasswordPolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi GetAccountPasswordPolicy AWS SDK for JavaScriptAPIReference.

`GetPolicy`

Il seguente esempio di codice mostra come utilizzare`GetPolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi GetPolicy AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi GetPolicy AWS SDK for JavaScriptAPIReference.

`GetRole`

Il seguente esempio di codice mostra come utilizzare`GetRole`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi GetRole AWS SDK for JavaScriptAPIReference.

`GetServerCertificate`

Il seguente esempio di codice mostra come utilizzare`GetServerCertificate`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi GetServerCertificate AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi GetServerCertificate AWS SDK for JavaScriptAPIReference.

`GetServiceLinkedRoleDeletionStatus`

Il seguente esempio di codice mostra come utilizzare`GetServiceLinkedRoleDeletionStatus`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 API i dettagli, vedi GetServiceLinkedRoleDeletionStatus AWS SDK for JavaScriptAPIReference.

`ListAccessKeys`

Il seguente esempio di codice mostra come utilizzare`ListAccessKeys`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListAccessKeys AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi ListAccessKeys AWS SDK for JavaScriptAPIReference.

`ListAccountAliases`

Il seguente esempio di codice mostra come utilizzare`ListAccountAliases`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListAccountAliases AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi ListAccountAliases AWS SDK for JavaScriptAPIReference.

`ListAttachedRolePolicies`

Il seguente esempio di codice mostra come utilizzare`ListAttachedRolePolicies`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListAttachedRolePolicies AWS SDK for JavaScriptAPIReference.

`ListGroups`

Il seguente esempio di codice mostra come utilizzare`ListGroups`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListGroups AWS SDK for JavaScriptAPIReference.

`ListPolicies`

Il seguente esempio di codice mostra come utilizzare`ListPolicies`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListPolicies AWS SDK for JavaScriptAPIReference.

`ListRolePolicies`

Il seguente esempio di codice mostra come utilizzare`ListRolePolicies`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListRolePolicies AWS SDK for JavaScriptAPIReference.

`ListRoles`

Il seguente esempio di codice mostra come utilizzare`ListRoles`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListRoles AWS SDK for JavaScriptAPIReference.

`ListSAMLProviders`

Il seguente esempio di codice mostra come utilizzare`ListSAMLProviders`.

SDKper JavaScript (v3)

> **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 i SAML fornitori.

```
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 API i dettagli, vedere L istSAMLProviders in AWS SDK for JavaScript APIReference.

`ListServerCertificates`

Il seguente esempio di codice mostra come utilizzare`ListServerCertificates`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListServerCertificates AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi ListServerCertificates AWS SDK for JavaScriptAPIReference.

`ListUsers`

Il seguente esempio di codice mostra come utilizzare`ListUsers`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListUsers AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi ListUsers AWS SDK for JavaScriptAPIReference.

`PutRolePolicy`

Il seguente esempio di codice mostra come utilizzare`PutRolePolicy`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi PutRolePolicy AWS SDK for JavaScriptAPIReference.

`UpdateAccessKey`

Il seguente esempio di codice mostra come utilizzare`UpdateAccessKey`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi UpdateAccessKey AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi UpdateAccessKey AWS SDK for JavaScriptAPIReference.

`UpdateServerCertificate`

Il seguente esempio di codice mostra come utilizzare`UpdateServerCertificate`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi UpdateServerCertificate AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi UpdateServerCertificate AWS SDK for JavaScriptAPIReference.

`UpdateUser`

Il seguente esempio di codice mostra come utilizzare`UpdateUser`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi UpdateUser AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi UpdateUser AWS SDK for JavaScriptAPIReference.

`UploadServerCertificate`

Il seguente esempio di codice mostra come utilizzare`UploadServerCertificate`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi UploadServerCertificate AWS SDK for JavaScriptAPIReference.

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 Amazon EC2 Auto Scaling per creare istanze Amazon Elastic Compute Cloud (AmazonEC2) basate su un modello di avvio e per mantenere il numero di istanze in un intervallo specificato.
- Gestisci e distribuisci HTTP le richieste 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 EC2 istanza 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 di integrità aggiornando AWS Systems Manager i parametri.

SDKper JavaScript (v3)

> **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.

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.

```
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 { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 () => {
    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,
        },
      }),
    );
  }),
  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) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
```

```
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    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,
      }),
    );
    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) => {
    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] },
    );
  }),
  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) => {
    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 },
        ],
      }),
    );
    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) => {
```

```
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  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,
      );
    }
```

```
  }),
  saveState,
];
```

Crea i passaggi per eseguire la demo.

```
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) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
```

```
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
```

```
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
```

```
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
```

```
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );

      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",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Crea i passaggi per distruggere tutte le risorse.

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
```

```
} 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 { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.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 = [
  loadState,
  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 {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } 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 {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } 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 {
      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.");
        }
      });
    } 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) => {
  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;
  }
}),
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,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
```

```
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - 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 utilizzate IAM gli utenti per l'autenticazione quando sviluppate software appositamente creato o lavorate 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.

SDKper JavaScript (v3)

> **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.

Crea un IAM utente e un ruolo che concedano l'autorizzazione a 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,
  GetUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
```

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
```

```
      return User;
    } else {
      throw caught;
    }
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
 STS).
  // It's not best practice to use access keys. For more information, see https://
aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;

  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
```

```
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
      return await listBuckets(s3Client);
    } catch (err) {
      if (err instanceof Error && err.name === "InvalidAccessKeyId") {
        throw err;
      }
    }
  });

  // Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
  // is thrown while the user and access keys are still stabilizing.
  const { Role } = await retry(
    {
      intervalInMs: 2000,
      maxRetries: 60,
    },
    () =>
      iamClient.send(
        new CreateRoleCommand({
          AssumeRolePolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
              {
                Effect: "Allow",
                Principal: {
                  // Allow the previously created user to assume this role.
                  AWS: User.Arn,
                },
                Action: "sts:AssumeRole",
              },
            ],
          }),
          RoleName: roleName,
        }),
      ),
  );

  if (!Role) {
    throw new Error("Role not created");
  }

  // Create a policy that allows the user to list S3 buckets.
```

```
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["s3:ListAllMyBuckets"],
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  }),
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
```

```
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

await iamClient.send(
  new DeletePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
  }),
);

await iamClient.send(
  new DeleteRoleCommand({
    RoleName: Role.RoleName,
  }),
```

```
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- Per API i dettagli, consulta i seguenti argomenti in Reference.AWS SDK for JavaScript API
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy

- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Esempi di Kinesis 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 Kinesis.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Esempi serverless

Esempi serverless

Richiamare una funzione Lambda da un trigger Kinesis

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Kinesis con Lambda utilizzando. JavaScript

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

```
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Consumo di un evento Kinesis con Lambda utilizzando. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
```

```
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Segnalazione degli errori degli elementi batch di Kinesis utilizzando Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";
```

```
const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Esempi di Lambda 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 Lambda.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui 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.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListFunctions AWS SDK for JavaScriptAPIReference.

Argomenti

- Azioni
- Scenari
- Esempi serverless

Azioni

`CreateFunction`

Il seguente esempio di codice mostra come utilizzare`CreateFunction`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi CreateFunction AWS SDK for JavaScriptAPIReference.

`DeleteFunction`

Il seguente esempio di codice mostra come utilizzare`DeleteFunction`.

SDKper JavaScript (v3)

> **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.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per API i dettagli, vedi DeleteFunction AWS SDK for JavaScriptAPIReference.

`GetFunction`

Il seguente esempio di codice mostra come utilizzare`GetFunction`.

SDKper JavaScript (v3)

> **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.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per API i dettagli, vedi GetFunction AWS SDK for JavaScriptAPIReference.

`Invoke`

Il seguente esempio di codice mostra come utilizzare`Invoke`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, consulta Invoke in AWS SDK for JavaScript APIReference.

`ListFunctions`

Il seguente esempio di codice mostra come usare`ListFunctions`.

SDKper JavaScript (v3)

> **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.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Per API i dettagli, vedi ListFunctions AWS SDK for JavaScriptAPIReference.

`UpdateFunctionCode`

Il seguente esempio di codice mostra come utilizzare`UpdateFunctionCode`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi UpdateFunctionCode AWS SDK for JavaScriptAPIReference.

`UpdateFunctionConfiguration`

Il seguente esempio di codice mostra come utilizzare`UpdateFunctionConfiguration`.

SDKper JavaScript (v3)

> **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.

```
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 API i dettagli, vedi UpdateFunctionConfiguration AWS SDK for JavaScriptAPIReference.

Scenari

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDKper 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, vedi l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- APIGateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Crea un'applicazione per analizzare il feedback dei clienti

L'esempio di codice seguente mostra come creare un'applicazione che analizza le schede dei commenti dei clienti, le traduce dalla loro lingua originale, ne determina il sentiment e genera un file audio dal testo tradotto.

SDKper 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

Nozioni di base sulle funzioni

L'esempio di codice seguente mostra come:

- Crea un IAM ruolo 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.

SDKper JavaScript (v3)

> **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.

Crea un ruolo AWS Identity and Access Management (IAM) che conceda a Lambda il permesso di scrivere 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 IAM ruolo 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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIReference.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Richiamo a una funzione Lambda da un browser

Il seguente esempio di codice mostra come richiamare una AWS Lambda funzione da un browser.

SDKper 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

SDKper 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

Usa API Gateway per richiamare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da Amazon API Gateway.

SDKper JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando il runtime Lambda JavaScript . API Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare una funzione Lambda richiamata da API Amazon Gateway che scansiona una tabella Amazon DynamoDB per gli anniversari di lavoro e utilizza Amazon Simple SNS Notification Service (Amazon) per inviare un messaggio di testo ai dipendenti in cui si congratula per la data del primo anniversario.

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

- APIGateway
- DynamoDB
- Lambda
- Amazon SNS

Utilizzo degli eventi pianificati per richiamare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

SDKper 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, si crea una funzione Lambda utilizzando il runtime Lambda. JavaScript API 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

Esempi serverless

Connessione a un RDS database Amazon in una funzione Lambda

Il seguente esempio di codice mostra come implementare una funzione Lambda che si connette a un RDS database. La funzione effettua una semplice richiesta al database e restituisce il risultato.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un RDS database Amazon in una funzione Lambda utilizzando. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
```

```
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

Connessione a un RDS database Amazon in una funzione Lambda utilizzando. TypeScript

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
 DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });
```

```
    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
 string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

Richiamare una funzione Lambda da un trigger Kinesis

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Kinesis con Lambda utilizzando. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Consumo di un evento Kinesis con Lambda utilizzando. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Richiama una funzione Lambda da un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DynamoDB e registra il contenuto del record.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento DynamoDB con Lambda utilizzando. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consumo di un evento DynamoDB con Lambda utilizzando. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
```

```
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Richiama una funzione Lambda da un trigger di Amazon DocumentDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso di modifiche di DocumentDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

SDKper (v3) JavaScript

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Amazon DocumentDB con Lambda utilizzando. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

Consumo di un evento Amazon DocumentDB con Lambda utilizzando TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

Richiama una funzione Lambda da un trigger Amazon MSK

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un cluster Amazon. MSK La funzione recupera il MSK payload e registra il contenuto del record.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumazione di un MSK evento Amazon con Lambda utilizzando. JavaScript

```
exports.handler = async (event) => {
```

```
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
      })
    }
}
```

Richiamo di una funzione Lambda da un trigger Amazon S3

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro dell'evento e chiama Amazon S3 API per recuperare e registrare il tipo di contenuto dell'oggetto.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento S3 con JavaScript Lambda utilizzando.

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

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
```

```
    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;

    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
        console.log(message);
        throw new Error(message);
    }
};
```

Consumo di un evento S3 con TypeScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
```

```
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

Richiama una funzione Lambda da un trigger Amazon SNS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da un argomento. SNS La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumare un SNS evento utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
```

```
}
```

Consumare un SNS evento utilizzando Lambda. TypeScript

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

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Richiama una funzione Lambda da un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda. SQS La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumare un SQS evento utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumare un SQS evento utilizzando Lambda. TypeScript

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

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
```

```
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
```

```
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Segnalazione degli errori degli elementi batch di Kinesis utilizzando Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
```

```
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Segnalazione degli errori degli elementi batch per le funzioni Lambda con un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando Lambda. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando Lambda. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;
```

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

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

Segnalazione degli errori degli articoli in batch per le funzioni Lambda con un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da una SQS coda. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi del SQS batch con JavaScript Lambda utilizzando.

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
```

```
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

Segnalazione degli errori degli elementi del SQS batch con TypeScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

Esempi di Amazon Lex 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 Lex.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un chatbot Amazon Lex

Il seguente esempio di codice mostra come creare un chatbot per coinvolgere i visitatori del tuo sito web.

SDKper JavaScript (v3)

> Mostra come utilizzare Amazon Lex API 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

MSKEsempi di Amazon 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 AmazonMSK.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Esempi serverless

Richiama una funzione Lambda da un trigger Amazon MSK

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un cluster Amazon. MSK La funzione recupera il MSK payload e registra il contenuto del record.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumazione di un MSK evento Amazon con Lambda utilizzando. JavaScript

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
```

```
      })
    }
}
```

Esempi di Amazon Personalize usando 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. Mentre le azioni mostrano come chiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`CreateBatchInferenceJob`

Il seguente esempio di codice mostra come utilizzare`CreateBatchInferenceJob`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateBatchInferenceJob AWS SDK for JavaScriptAPIReference.

`CreateBatchSegmentJob`

Il seguente esempio di codice mostra come utilizzare`CreateBatchSegmentJob`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateBatchSegmentJob AWS SDK for JavaScriptAPIReference.

`CreateCampaign`

Il seguente esempio di codice mostra come utilizzare`CreateCampaign`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateCampaign AWS SDK for JavaScriptAPIReference.

`CreateDataset`

Il seguente esempio di codice mostra come utilizzare`CreateDataset`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateDataset AWS SDK for JavaScriptAPIReference.

`CreateDatasetExportJob`

Il seguente esempio di codice mostra come utilizzare`CreateDatasetExportJob`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateDatasetExportJob AWS SDK for JavaScriptAPIReference.

`CreateDatasetGroup`

Il seguente esempio di codice mostra come utilizzare`CreateDatasetGroup`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedere CreateDatasetGroupin AWS SDK for JavaScript APIReference.

`CreateDatasetImportJob`

Il seguente esempio di codice mostra come utilizzare`CreateDatasetImportJob`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateDatasetImportJob AWS SDK for JavaScriptAPIReference.

`CreateEventTracker`

Il seguente esempio di codice mostra come utilizzare`CreateEventTracker`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateEventTracker AWS SDK for JavaScriptAPIReference.

`CreateFilter`

Il seguente esempio di codice mostra come utilizzare`CreateFilter`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateFilter AWS SDK for JavaScriptAPIReference.

`CreateRecommender`

Il seguente esempio di codice mostra come utilizzare`CreateRecommender`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateRecommender AWS SDK for JavaScriptAPIReference.

`CreateSchema`

Il seguente esempio di codice mostra come utilizzare`CreateSchema`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema /* required */
};
```

```
export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Crea uno schema con un dominio.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema, /* required */
  domain: 'DOMAIN'   /* required for a domain dataset group, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
};

export const run = async () => {
```

```
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createDomainSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per API i dettagli, vedere CreateSchemain AWS SDK for JavaScript APIReference.

`CreateSolution`

Il seguente esempio di codice mostra come utilizzare`CreateSolution`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateSolution AWS SDK for JavaScriptAPIReference.

`CreateSolutionVersion`

Il seguente esempio di codice mostra come utilizzare`CreateSolutionVersion`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi CreateSolutionVersion AWS SDK for JavaScriptAPIReference.

Esempi di Amazon Personalize Events 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 Personalize Events.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Mentre le azioni mostrano come chiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`PutEvents`

Il seguente esempio di codice mostra come utilizzare`PutEvents`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi PutEvents AWS SDK for JavaScriptAPIReference.

`PutItems`

Il seguente esempio di codice mostra come utilizzare`PutItems`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi PutItems AWS SDK for JavaScriptAPIReference.

`PutUsers`

Il seguente esempio di codice mostra come utilizzare`PutUsers`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi PutUsers AWS SDK for JavaScriptAPIReference.

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. Mentre le azioni mostrano come chiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`GetPersonalizedRanking`

Il seguente esempio di codice mostra come utilizzare`GetPersonalizedRanking`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API i dettagli, vedi GetPersonalizedRanking AWS SDK for JavaScriptAPIReference.

`GetRecommendations`

Il seguente esempio di codice mostra come utilizzare`GetRecommendations`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get 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 API dettagli, consulta Reference GetRecommendations.AWS SDK for JavaScript API

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. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`SendMessages`

Il seguente esempio di codice mostra come utilizzare`SendMessages`.

SDKper JavaScript (v3)

> **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.

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";
export const pinClient = new PinpointClient({ region: REGION });
```

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 { MessageResponse } = await pinClient.send(
      new SendMessagesCommand(params),
    );

    if (!MessageResponse) {
      throw new Error("No message response.");
    }

    if (!MessageResponse.Result) {
      throw new Error("No message result.");
```

```
    }

    const recipientResult = MessageResponse.Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    } else {
      console.log(recipientResult.MessageId);
    }
  } catch (err) {
    console.log(err.message);
  }
};

run();
```

Manda un SMS messaggio.

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

/* The phone number or short code to send the message from. The phone number
 or short code that you specify has to be associated with your Amazon Pinpoint
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number.  For best results, you should specify the phone
 number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXX
```

```
/* The type of SMS message that you want to send. If you plan to send
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderID";

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"],
    );
  } catch (err) {
```

```
    console.log(err);
  }
};
run();
```

- Per API i dettagli, vedere SendMessagesin AWS SDK for JavaScript APIReference.

SDKper 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"]
    );
  }
});
```

Manda un SMS messaggio.

```
"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 API i dettagli, vedere SendMessagesin AWS SDK for JavaScript APIReference.

Esempi di Amazon Polly 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 Polly.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Crea un'applicazione per analizzare il feedback dei clienti

L'esempio di codice seguente mostra come creare un'applicazione che analizza le schede dei commenti dei clienti, le traduce dalla loro lingua originale, ne determina il sentiment e genera un file audio dal testo tradotto.

SDKper 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

RDSEsempi di Amazon 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 AmazonRDS.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

Il seguente esempio di codice mostra come creare un'applicazione Web che tiene traccia degli elementi di lavoro in un database Amazon Aurora Serverless e utilizza Amazon Simple Email Service (AmazonSES) per inviare report.

SDKper 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 (AmazonSES). 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. AWS servizi
- Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
- Invia un report via e-mail sugli elementi di lavoro filtrati utilizzando AmazonSES.
- Distribuisci e gestisci risorse di esempio con lo script incluso. AWS CloudFormation

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
- Servizio RDS dati Amazon
- Amazon SES

Esempi serverless

Connessione a un RDS database Amazon in una funzione Lambda

Il seguente esempio di codice mostra come implementare una funzione Lambda che si connette a un RDS database. La funzione effettua una semplice richiesta al database e restituisce il risultato.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un RDS database Amazon in una funzione Lambda utilizzando. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
```

```
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

Connessione a un RDS database Amazon in una funzione Lambda utilizzando. TypeScript

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
 DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {
```

```
    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
 string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
```

```
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

Esempi di Amazon RDS Data Service 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 RDS Data Service.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Scenari

Scenari

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

Il seguente esempio di codice mostra come creare un'applicazione Web che tiene traccia degli elementi di lavoro in un database Amazon Aurora Serverless e utilizza Amazon Simple Email Service (AmazonSES) per inviare report.

SDKper 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 (AmazonSES). 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. AWS servizi

- Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
- Invia un report via e-mail sugli elementi di lavoro filtrati utilizzando AmazonSES.
- Distribuisci e gestisci risorse di esempio con lo script incluso. AWS CloudFormation

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
- Servizio RDS dati Amazon
- Amazon SES

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. Mentre le azioni mostrano come chiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`CreateCluster`

Il seguente esempio di codice mostra come utilizzare`CreateCluster`.

SDKper JavaScript (v3)

> **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.

Crea il client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

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 API i dettagli, vedi CreateCluster AWS SDK for JavaScriptAPIReference.

`DeleteCluster`

Il seguente esempio di codice mostra come utilizzare`DeleteCluster`.

SDKper JavaScript (v3)

> **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.

Crea il client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

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 API i dettagli, vedi DeleteCluster AWS SDK for JavaScriptAPIReference.

`DescribeClusters`

Il seguente esempio di codice mostra come utilizzare`DescribeClusters`.

SDKper JavaScript (v3)

> **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.

Crea il client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

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 API i dettagli, consulta la sezione DescribeClusters AWS SDK for JavaScriptAPIReference.

`ModifyCluster`

Il seguente esempio di codice mostra come utilizzare`ModifyCluster`.

SDKper JavaScript (v3)

> **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.

Crea il client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
```

```
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

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 API i dettagli, vedere ModifyClusterin AWS SDK for JavaScript APIReference.

Esempi di Amazon Rekognition SDK con for (v3) JavaScript

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Rekognition.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDKper 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, vedi l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- APIGateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rileva PPE nelle immagini

Il seguente esempio di codice mostra come creare un'app che utilizza Amazon Rekognition per rilevare i dispositivi di protezione PPE individuale () nelle immagini.

SDKper JavaScript (v3)

Mostra come usare Amazon Rekognition AWS SDK for JavaScript con per creare un'applicazione per rilevare dispositivi di protezione PPE individuale () nelle immagini che si trovano in un bucket Amazon Simple Storage Service (Amazon S3). L'app salva i risultati in una tabella Amazon DynamoDB e invia all'amministratore una notifica e-mail con i risultati utilizzando Amazon Simple Email Service (Amazon). SES

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per PPE usare Amazon Rekognition.
- Verifica un indirizzo email per AmazonSES.
- Aggiornare una tabella DynamoDB con i risultati.
- Invia una notifica via e-mail tramite AmazonSES.

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 Rekognition
- Amazon S3
- Amazon SES

Rilevamento di oggetti nelle immagini

Il seguente esempio di codice mostra come creare un'app che utilizza Amazon Rekognition per rilevare oggetti per categoria nelle immagini.

SDKper JavaScript (v3)

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'app invia all'amministratore una notifica e-mail con i risultati utilizzando Amazon Simple Email Service (AmazonSES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.

- Analizza le immagini per rilevare gli oggetti tramite Amazon Rekognition.
- Verifica un indirizzo email per AmazonSES.
- Invia una notifica via e-mail tramite AmazonSES.

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 Rekognition
- Amazon S3
- Amazon SES

Rilevamento di persone e oggetti in un video

Il seguente esempio di codice mostra come rilevare persone e oggetti in un video con Amazon Rekognition.

SDKper JavaScript (v3)

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'app invia all'amministratore una notifica e-mail con i risultati utilizzando Amazon Simple Email Service (AmazonSES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per PPE usare Amazon Rekognition.
- Verifica un indirizzo email per AmazonSES.
- Invia una notifica via e-mail tramite AmazonSES.

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 Rekognition
- Amazon S3
- Amazon SES

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 nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto nei relativi scenari.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile 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.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListBuckets AWS SDK for JavaScriptAPIReference.

Argomenti

- Nozioni di base
- Azioni
- Scenari
- Esempi serverless

Nozioni di base

Scopri le nozioni di base

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.

SDKper JavaScript (v3)

> **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.

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 SDK metodo 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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

Azioni

`CopyObject`

Il seguente esempio di codice mostra come usare`CopyObject`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi CopyObject AWS SDK for JavaScriptAPIReference.

`CreateBucket`

Il seguente esempio di codice mostra come utilizzare`CreateBucket`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi CreateBucket AWS SDK for JavaScriptAPIReference.

`DeleteBucket`

Il seguente esempio di codice mostra come utilizzare`DeleteBucket`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteBucket AWS SDK for JavaScriptAPIReference.

`DeleteBucketPolicy`

Il seguente esempio di codice mostra come utilizzare`DeleteBucketPolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteBucketPolicy AWS SDK for JavaScriptAPIReference.

`DeleteBucketWebsite`

Il seguente esempio di codice mostra come utilizzare`DeleteBucketWebsite`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteBucketWebsite AWS SDK for JavaScriptAPIReference.

`DeleteObject`

Il seguente esempio di codice mostra come utilizzare`DeleteObject`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteObject AWS SDK for JavaScriptAPIReference.

`DeleteObjects`

Il seguente esempio di codice mostra come utilizzare`DeleteObjects`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DeleteObjects AWS SDK for JavaScriptAPIReference.

`GetBucketAcl`

Il seguente esempio di codice mostra come utilizzare`GetBucketAcl`.

SDKper JavaScript (v3)

> **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.

Ottieni le ACL autorizzazioni.

```
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 API i dettagli, consulta la sezione GetBucketAcl AWS SDK for JavaScriptAPIReference.

`GetBucketCors`

Il seguente esempio di codice mostra come utilizzare`GetBucketCors`.

SDKper JavaScript (v3)

> **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.

Ottieni la CORS polizza per il secchio.

```
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 API i dettagli, consulta la sezione GetBucketCors AWS SDK for JavaScriptAPIReference.

`GetBucketPolicy`

Il seguente esempio di codice mostra come utilizzare`GetBucketPolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi GetBucketPolicy AWS SDK for JavaScriptAPIReference.

`GetBucketWebsite`

Il seguente esempio di codice mostra come utilizzare`GetBucketWebsite`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi GetBucketWebsite AWS SDK for JavaScriptAPIReference.

`GetObject`

Il seguente esempio di codice mostra come utilizzare`GetObject`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi GetObject AWS SDK for JavaScriptAPIReference.

`GetObjectLockConfiguration`

Il seguente esempio di codice mostra come utilizzare`GetObjectLockConfiguration`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new GetObjectLockConfigurationCommand({
    Bucket: bucketName,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
  });

  try {
    const { ObjectLockConfiguration } = await client.send(command);
    console.log(`Object Lock Configuration: ${ObjectLockConfiguration}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

- Per API i dettagli, vedi GetObjectLockConfiguration AWS SDK for JavaScriptAPIReference.

`GetObjectRetention`

Il seguente esempio di codice mostra come utilizzare`GetObjectRetention`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const { Retention } = await client.send(command);
    console.log(`Object Retention Settings: ${Retention.Status}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Per API i dettagli, vedi GetObjectRetention AWS SDK for JavaScriptAPIReference.

`ListBuckets`

Il seguente esempio di codice mostra come utilizzare`ListBuckets`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedi ListBuckets AWS SDK for JavaScriptAPIReference.

`ListObjectsV2`

Il seguente esempio di codice mostra come utilizzare`ListObjectsV2`.

SDKper JavaScript (v3)

> **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 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 API i dettagli, vedere ListObjectsV2 in AWS SDK for JavaScript API Reference.

`PutBucketAcl`

Il seguente esempio di codice mostra come utilizzare`PutBucketAcl`.

SDKper JavaScript (v3)

> **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.

Metti il secchio. ACL

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

const client = new S3Client({});

// Most Amazon S3 use cases don't require the use of access control lists (ACLs).
// We recommend that you disable ACLs, except in unusual circumstances where
// you need to control access for each object individually.
// Consider a policy instead. For more information see https://docs.aws.amazon.com/
AmazonS3/latest/userguide/bucket-policies.html.
export const main = async () => {
  // Grant a user READ access to a bucket.
  const command = new PutBucketAclCommand({
    Bucket: "test-bucket",
    AccessControlPolicy: {
```

```
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
 AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
            ID: "canonical-id-1",
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "FULL_CONTROL",
        },
      ],
      Owner: {
        ID: "canonical-id-2",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per API i dettagli, vedere PutBucketAclin AWS SDK for JavaScript APIReference.

`PutBucketCors`

Il seguente esempio di codice mostra come utilizzare`PutBucketCors`.

SDKper JavaScript (v3)

> **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.

Aggiungi una CORS regola.

```
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 API i dettagli, PutBucketCorsconsulta AWS SDK for JavaScript APIReference.

`PutBucketPolicy`

Il seguente esempio di codice mostra come utilizzare`PutBucketPolicy`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi PutBucketPolicy AWS SDK for JavaScriptAPIReference.

`PutBucketWebsite`

Il seguente esempio di codice mostra come utilizzare`PutBucketWebsite`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi PutBucketWebsite AWS SDK for JavaScriptAPIReference.

`PutObject`

Il seguente esempio di codice mostra come utilizzare`PutObject`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi PutObject AWS SDK for JavaScriptAPIReference.

`PutObjectLegalHold`

Il seguente esempio di codice mostra come utilizzare`PutObjectLegalHold`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
```

```
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    LegalHold: {
      // Set the status to 'ON' to place a legal hold on the object.
      // Set the status to 'OFF' to remove the legal hold.
      Status: "ON",
    },
    // Optionally, you can provide additional parameters
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object legal hold status: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Per API i dettagli, vedi PutObjectLegalHold AWS SDK for JavaScriptAPIReference.

`PutObjectLockConfiguration`

Il seguente esempio di codice mostra come utilizzare`PutObjectLockConfiguration`.

SDKper JavaScript (v3)

> **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.

Imposta la configurazione di blocco degli oggetti di un bucket.

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Lock Configuration updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

```
// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

Imposta il periodo di conservazione predefinito di un bucket.

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
      Rule: {
        DefaultRetention: {
          Mode: "GOVERNANCE",
          Years: 3,
        },
      },
    },
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Default Object Lock Configuration updated: ${response.
$metadata.httpStatusCode}`,
    );
```

```
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

- Per API i dettagli, vedere PutObjectLockConfigurationin AWS SDK for JavaScript APIReference.

`PutObjectRetention`

Il seguente esempio di codice mostra come utilizzare`PutObjectRetention`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    BypassGovernanceRetention: false,
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
```

```
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    Retention: {
      Mode: "GOVERNANCE", // or "COMPLIANCE"
      RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Retention settings updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Per API i dettagli, vedi PutObjectRetention AWS SDK for JavaScriptAPIReference.

Scenari

Crea un predefinito URL

Il seguente esempio di codice mostra come creare un predefinito URL per Amazon S3 e caricare un oggetto.

SDKper JavaScript (v3)

> **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.

Crea un predefinito URL 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 predefinito URL 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.

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDKper 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, vedi l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- APIGateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

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.

SDKper JavaScript (v3)

> **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.

Il codice seguente è il componente React pertinente che effettua chiamate a 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 API dettagli, vedere ListObjectsin Reference.AWS SDK for JavaScript API

Creazione di un'applicazione Amazon Textract explorer

Il seguente esempio di codice mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDKper 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 Amazon Simple Queue Service (Amazon) sottoscritta a un argomento SQS Amazon Simple Notification Service (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

Rileva PPE nelle immagini

Il seguente esempio di codice mostra come creare un'app che utilizza Amazon Rekognition per rilevare i dispositivi di protezione PPE individuale () nelle immagini.

SDKper JavaScript (v3)

Mostra come usare Amazon Rekognition AWS SDK for JavaScript con per creare un'applicazione per rilevare dispositivi di protezione PPE individuale () nelle immagini che si trovano in un bucket Amazon Simple Storage Service (Amazon S3). L'app salva i risultati in una tabella Amazon DynamoDB e invia all'amministratore una notifica e-mail con i risultati utilizzando Amazon Simple Email Service (Amazon). SES

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.

- Analizza le immagini per PPE usare Amazon Rekognition.
- Verifica un indirizzo email per AmazonSES.
- Aggiornare una tabella DynamoDB con i risultati.
- Invia una notifica via e-mail tramite AmazonSES.

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 Rekognition
- Amazon S3
- Amazon SES

Rilevamento di oggetti nelle immagini

Il seguente esempio di codice mostra come creare un'app che utilizza Amazon Rekognition per rilevare oggetti per categoria nelle immagini.

SDKper JavaScript (v3)

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'app invia all'amministratore una notifica e-mail con i risultati utilizzando Amazon Simple Email Service (AmazonSES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per rilevare gli oggetti tramite Amazon Rekognition.
- Verifica un indirizzo email per AmazonSES.
- Invia una notifica via e-mail tramite AmazonSES.

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 Rekognition

- Amazon S3
- Amazon SES

Ottieni la configurazione di conservazione legale di un oggetto

Il seguente esempio di codice mostra come ottenere la configurazione di conservazione legale di un bucket S3.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
  } catch (err) {
    console.error(err);
  }
```

```
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "DOC-EXAMPLE-BUCKET", "OBJECT_KEY");
}
```

- Per API i dettagli, vedi GetObjectLegalHold AWS SDK for JavaScriptAPIReference.

Blocca oggetti Amazon S3

Il seguente esempio di codice mostra come utilizzare le funzionalità di blocco degli oggetti di S3.

SDKper JavaScript (v3)

> **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.

index.js- Punto di ingresso per il flusso di lavoro. Questo orchestra tutti i passaggi. Visita GitHub per vedere i dettagli di implementazione di Scenario, ScenarioInput ScenarioOutput, e. ScenarioAction

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "./welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  confirmSetLegalHoldFileEnabled,
  confirmSetLegalHoldFileRetention,
  confirmSetRetentionPeriodFileEnabled,
  confirmSetRetentionPeriodFileRetention,
  confirmUpdateLockPolicy,
```

```
  confirmUpdateRetention,
  createBuckets,
  createBucketsAction,
  populateBuckets,
  populateBucketsAction,
  setLegalHoldFileEnabledAction,
  setLegalHoldFileRetentionAction,
  setRetentionPeriodFileEnabledAction,
  setRetentionPeriodFileRetentionAction,
  updateLockPolicy,
  updateLockPolicyAction,
  updateRetention,
  updateRetentionAction,
} from "./setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
    deploy: new scenarios.Scenario(
      "S3 Object Locking - Deploy",
      [
        welcome(scenarios),
        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
        createBucketsAction(scenarios, client),
        updateRetention(scenarios),
        confirmUpdateRetention(scenarios),
        exitOnFalse(scenarios, "confirmUpdateRetention"),
        updateRetentionAction(scenarios, client),
        populateBuckets(scenarios),
        confirmPopulateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        updateLockPolicy(scenarios),
        confirmUpdateLockPolicy(scenarios),
        exitOnFalse(scenarios, "confirmUpdateLockPolicy"),
```

```
        updateLockPolicyAction(scenarios, client),
        confirmSetLegalHoldFileEnabled(scenarios),
        setLegalHoldFileEnabledAction(scenarios, client),
        confirmSetRetentionPeriodFileEnabled(scenarios),
        setRetentionPeriodFileEnabledAction(scenarios, client),
        confirmSetLegalHoldFileRetention(scenarios),
        setLegalHoldFileRetentionAction(scenarios, client),
        confirmSetRetentionPeriodFileRetention(scenarios),
        setRetentionPeriodFileRetentionAction(scenarios, client),
        saveState,
      ],
      initialState,
    ),
    demo: new scenarios.Scenario(
      "S3 Object Locking - Demo",
      [loadState, replAction(scenarios, client)],
      initialState,
    ),
    clean: new scenarios.Scenario(
      "S3 Object Locking - Destroy",
      [
        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
      ],
      initialState,
    ),
  };
};

// Call function if run directly
import { fileURLToPath } from "url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios);
}
```

welcome.steps.js- Invia messaggi di benvenuto alla console.

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    `Welcome to the Amazon Simple Storage Service (S3) Object Locking Workflow
 Scenario. For this workflow, we will use the AWS SDK for JavaScript to create
 several S3 buckets and files to demonstrate working with S3 locking features.`,
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };
```

setup.steps.js- Distribuisci bucket, oggetti e impostazioni dei file.

```
import {
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
```

```
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const bucketPrefix = "js-object-locking";

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    `The following buckets will be created:
         ${bucketPrefix}-no-lock with object lock False.
         ${bucketPrefix}-lock-enabled with object lock True.
         ${bucketPrefix}-retention-after-creation with object lock False.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const noLockBucketName = `${bucketPrefix}-no-lock`;
    const lockEnabledBucketName = `${bucketPrefix}-lock-enabled`;
```

```
    const retentionBucketName = `${bucketPrefix}-retention-after-creation`;

    await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
    await client.send(
      new CreateBucketCommand({
        Bucket: lockEnabledBucketName,
        ObjectLockEnabledForBucket: true,
      }),
    );
    await client.send(new CreateBucketCommand({ Bucket: retentionBucketName }));

    state.noLockBucketName = noLockBucketName;
    state.lockEnabledBucketName = lockEnabledBucketName;
    state.retentionBucketName = retentionBucketName;
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    `The following test files will be created:
         file0.txt in ${bucketPrefix}-no-lock.
         file1.txt in ${bucketPrefix}-no-lock.
         file0.txt in ${bucketPrefix}-lock-enabled.
         file1.txt in ${bucketPrefix}-lock-enabled.
         file0.txt in ${bucketPrefix}-retention-after-creation.
         file1.txt in ${bucketPrefix}-retention-after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
```

```
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
```

```
    await client.send(
      new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateRetention = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateRetention",
    `A bucket can be configured to use object locking with a default retention
 period.
   A default retention period will be configured for ${bucketPrefix}-retention-
after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateRetention",
    "Configure default retention period?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateRetentionAction", async (state) => {
    await client.send(
      new PutBucketVersioningCommand({
        Bucket: state.retentionBucketName,
        VersioningConfiguration: {
          MFADelete: MFADeleteStatus.Disabled,
```

```
          Status: BucketVersioningStatus.Enabled,
        },
      }),
    );

    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.retentionBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
          Rule: {
            DefaultRetention: {
              Mode: "GOVERNANCE",
              Years: 1,
            },
          },
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateLockPolicy = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateLockPolicy",
    `Object lock policies can also be added to existing buckets.
   An object lock policy will be added to ${bucketPrefix}-lock-enabled.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateLockPolicy = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateLockPolicy",
    "Add object lock policy?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
```

```
 * @param {S3Client} client
 */
const updateLockPolicyAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateLockPolicyAction", async (state) => {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.lockEnabledBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileEnabled",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.lockEnabledBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileEnabledAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
```

```
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.lockEnabledBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetRetentionPeriodFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileEnabled",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.lockEnabledBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileEnabledAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
```

```
        }),
      );
      console.log(
        `Set retention for file1.txt in ${state.lockEnabledBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.retentionBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileRetentionAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.retentionBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.retentionBucketName}.`,
```

```
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileRetention },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmSetRetentionPeriodFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileRetention",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.retentionBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileRetentionAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.retentionBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
          BypassGovernanceRetention: true,
        }),
      );
      console.log(
```

```
        `Set retention for file1.txt in ${state.retentionBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileRetention },
  );

export {
  createBuckets,
  confirmCreateBuckets,
  createBucketsAction,
  populateBuckets,
  confirmPopulateBuckets,
  populateBucketsAction,
  updateRetention,
  confirmUpdateRetention,
  updateRetentionAction,
  updateLockPolicy,
  confirmUpdateLockPolicy,
  updateLockPolicyAction,
  confirmSetLegalHoldFileEnabled,
  setLegalHoldFileEnabledAction,
  confirmSetRetentionPeriodFileEnabled,
  setRetentionPeriodFileEnabledAction,
  confirmSetLegalHoldFileRetention,
  setLegalHoldFileRetentionAction,
  confirmSetRetentionPeriodFileRetention,
  setRetentionPeriodFileRetentionAction,
};
```

repl.steps.js- Visualizza ed elimina i file nei bucket.

```
import {
  ChecksumAlgorithm,
  DeleteObjectCommand,
  GetObjectLegalHoldCommand,
  GetObjectLockConfigurationCommand,
  GetObjectRetentionCommand,
  ListObjectVersionsCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";
```

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  DELETE_FILE: 2,
  DELETE_FILE_WITH_RETENTION: 3,
  OVERWRITE_FILE: 4,
  VIEW_RETENTION_SETTINGS: 5,
  VIEW_LEGAL_HOLD_SETTINGS: 6,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new scenarios.ScenarioInput(
    "replChoice",
    `Explore the S3 locking features by selecting one of the following choices`,
    {
      type: "select",
      choices: [
        { name: "List all files in buckets", value: choices.LIST_ALL_FILES },
        { name: "Attempt to delete a file.", value: choices.DELETE_FILE },
        {
          name: "Attempt to delete a file with retention period bypass.",
          value: choices.DELETE_FILE_WITH_RETENTION,
        },
        { name: "Attempt to overwrite a file.", value: choices.OVERWRITE_FILE },
        {
          name: "View the object and bucket retention settings for a file.",
          value: choices.VIEW_RETENTION_SETTINGS,
        },
        {
          name: "View the legal hold settings for a file.",
          value: choices.VIEW_LEGAL_HOLD_SETTINGS,
        },
        { name: "Finish the workflow.", value: choices.EXIT },
```

```
      ],
    },
  );

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;
      files.push({ bucket, key: Key, version: VersionId });
    }
  }

  return files;
};

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const replAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.noLockBucketName,
        state.lockEnabledBucketName,
        state.retentionBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file:",
        {
          type: "select",
          choices: files.map((file, index) => ({
```

```
            name: `${index + 1}: ${file.bucket}: ${file.key} (version: ${
              file.version
            })`,
            value: index,
          })),
        },
      );

      const { replChoice } = state;

      switch (replChoice) {
        case choices.LIST_ALL_FILES: {
          const files = await getAllFiles(client, [
            state.noLockBucketName,
            state.lockEnabledBucketName,
            state.retentionBucketName,
          ]);
          state.replOutput = files
            .map(
              (file) =>
                `${file.bucket}: ${file.key} (version: ${file.version})`,
            )
            .join("\n");
          break;
        }
        case choices.DELETE_FILE: {
          /** @type {number} */
          const fileToDelete = await fileInput.handle(state);
          const selectedFile = files[fileToDelete];
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
```

```
        }
        case choices.DELETE_FILE_WITH_RETENTION: {
          /** @type {number} */
          const fileToDelete = await fileInput.handle(state);
          const selectedFile = files[fileToDelete];
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
                BypassGovernanceRetention: true,
              }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.OVERWRITE_FILE: {
          /** @type {number} */
          const fileToOverwrite = await fileInput.handle(state);
          const selectedFile = files[fileToOverwrite];
          try {
            await client.send(
              new PutObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                Body: "New content",
                ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
              }),
            );
            state.replOutput = `Overwrote ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to overwrite object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.VIEW_RETENTION_SETTINGS: {
```

```
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
            const retention = await client.send(
              new GetObjectRetentionCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            const bucketConfig = await client.send(
              new GetObjectLockConfigurationCommand({
                Bucket: selectedFile.bucket,
              }),
            );
            state.replOutput = `Object retention for ${selectedFile.key}
 in ${selectedFile.bucket}: ${retention.Retention?.Mode} until
 ${retention.Retention?.RetainUntilDate?.toISOString()}.
Bucket object lock config for ${selectedFile.bucket} in ${selectedFile.bucket}:
Enabled: ${bucketConfig.ObjectLockConfiguration?.ObjectLockEnabled}
Rule:
 ${JSON.stringify(bucketConfig.ObjectLockConfiguration?.Rule?.DefaultRetention)}`;
          } catch (err) {
            state.replOutput = `Unable to fetch object lock retention:
 '${err.message}'`;
          }
          break;
        }
        case choices.VIEW_LEGAL_HOLD_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
            const legalHold = await client.send(
              new GetObjectLegalHoldCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Object legal hold for ${selectedFile.key} in
 ${selectedFile.bucket}: Status: ${legalHold.LegalHold?.Status}`;
          } catch (err) {
```

```
            state.replOutput = `Unable to fetch legal hold: '${err.message}'`;
          }
          break;
        }
        default:
          throw new Error(`Invalid replChoice: ${replChoice}`);
      }
    },
    {
      whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new scenarios.ScenarioOutput(
          "REPL output",
          (state) => state.replOutput,
          { preformatted: true },
        ),
      },
    },
  );

export { replInput, replAction, choices };
```

clean.steps.js- Distruggi tutte le risorse create.

```
import {
  DeleteObjectCommand,
  DeleteBucketCommand,
  ListObjectVersionsCommand,
  GetObjectLegalHoldCommand,
  GetObjectRetentionCommand,
  PutObjectLegalHoldCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */
```

```
/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { noLockBucketName, lockEnabledBucketName, retentionBucketName } =
      state;

    const buckets = [
      noLockBucketName,
      lockEnabledBucketName,
      retentionBucketName,
    ];

    for (const bucket of buckets) {
      /** @type {import("@aws-sdk/client-s3").ListObjectVersionsCommandOutput} */
      let objectsResponse;

      try {
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
        );
      } catch (e) {
        if (e instanceof Error && e.name === "NoSuchBucket") {
          console.log("Object's bucket has already been deleted.");
          continue;
        } else {
          throw e;
        }
      }

      for (const version of objectsResponse.Versions || []) {
        const { Key, VersionId } = version;
```

```
    try {
      const legalHold = await client.send(
        new GetObjectLegalHoldCommand({
          Bucket: bucket,
          Key,
          VersionId,
        }),
      );

      if (legalHold.LegalHold?.Status === "ON") {
        await client.send(
          new PutObjectLegalHoldCommand({
            Bucket: bucket,
            Key,
            VersionId,
            LegalHold: {
              Status: "OFF",
            },
          }),
        );
      }
    } catch (err) {
      console.log(
        `Unable to fetch legal hold for ${Key} in ${bucket}: '${err.message}'`,
      );
    }

    try {
      const retention = await client.send(
        new GetObjectRetentionCommand({
          Bucket: bucket,
          Key,
          VersionId,
        }),
      );

      if (retention.Retention?.Mode === "GOVERNANCE") {
        await client.send(
          new DeleteObjectCommand({
            Bucket: bucket,
            Key,
            VersionId,
            BypassGovernanceRetention: true,
```

```
              }),
            );
          }
        } catch (err) {
          console.log(
            `Unable to fetch object lock retention for ${Key} in ${bucket}:
'${err.message}'`,
          );
        }

        await client.send(
          new DeleteObjectCommand({
            Bucket: bucket,
            Key,
            VersionId,
          }),
        );
      }

      await client.send(new DeleteBucketCommand({ Bucket: bucket }));
      console.log(`Delete for ${bucket} complete.`);
    }
  });

export { confirmCleanup, cleanupAction };
```

- Per API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIReference.
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold
  - PutObjectLockConfiguration
  - PutObjectRetention

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.

SDKper JavaScript (v3)

> **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.

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);
};

/**
 * @param {string | undefined} contentRange
 */
export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
```

```
  const [start, end] = range.split("-");
  return {
    start: parseInt(start),
    end: parseInt(end),
    length: parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

// When downloading a large file, you might want to break it down into
// smaller pieces. Amazon S3 accepts a Range header to specify the start
// and end of the byte range to be downloaded.
const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url)),
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    console.log(`Downloading bytes ${nextRange.start} to ${nextRange.end}`);

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
    });

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

export const main = async () => {
  await downloadInChunks({
    bucket: "my-cool-bucket",
    key: "my-cool-object.txt",
  });
};
```

Esempi serverless

Richiamo di una funzione Lambda da un trigger Amazon S3

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro dell'evento e chiama Amazon S3 API per recuperare e registrare il tipo di contenuto dell'oggetto.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento S3 con JavaScript Lambda utilizzando.

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

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));

    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;
```

```
    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
        console.log(message);
        throw new Error(message);
    }
};
```

Consumo di un evento S3 con TypeScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

Esempi di S3 Glacier che utilizzano for (v3) SDK 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. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`CreateVault`

Il seguente esempio di codice mostra come utilizzare`CreateVault`.

SDKper JavaScript (v3)

> **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.

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 la 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 API i dettagli, vedi CreateVault AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi CreateVault AWS SDK for JavaScriptAPIReference.

`UploadArchive`

Il seguente esempio di codice mostra come utilizzare`UploadArchive`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi UploadArchive AWS SDK for JavaScriptAPIReference.

SDKper 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 API i dettagli, vedi UploadArchive AWS SDK for JavaScriptAPIReference.

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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve SageMaker

I seguenti esempi di codice mostrano come iniziare a utilizzare SageMaker.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 API i dettagli, vedi ListNotebookInstances AWS SDK for JavaScriptAPIReference.

Argomenti

- Azioni
- Scenari

Azioni

`CreatePipeline`

Il seguente esempio di codice mostra come utilizzare`CreatePipeline`.

SDKper JavaScript (v3)

> **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.

Una funzione che crea una SageMaker pipeline utilizzando una JSON definizione 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);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}
```

- Per API i dettagli, vedere CreatePipelinein AWS SDK for JavaScript APIReference.

`DeletePipeline`

Il seguente esempio di codice mostra come utilizzare`DeletePipeline`.

SDKper JavaScript (v3)

> **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.

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 API dettagli, vedere in Reference. DeletePipelineAWS SDK for JavaScript API

`DescribePipelineExecution`

Il seguente esempio di codice mostra come utilizzare`DescribePipelineExecution`.

SDKper JavaScript (v3)

> **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.

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, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- Per API i dettagli, consulta la sezione [DescribePipelineExecution AWS SDK for JavaScript](#)APIReference.

`StartPipelineExecution`

Il seguente esempio di codice mostra come utilizzare`StartPipelineExecution`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedere StartPipelineExecutionin AWS SDK for JavaScript APIReference.

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 AWS SDKs Community.aws.

SDKper (v3) 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.

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,
  GetRoleCommand,
  ListPoliciesCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
  GetFunctionCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
```

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

import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-
geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
  GetQueueUrlCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 props
 */
export async function createLambdaExecutionRole({ name, iamClient }) {
  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: { Service: ["lambda.amazonaws.com"] },
            },
          ],
        }),
```

```
      }),
    );

  let role = null;

  try {
    const { Role } = await createRole();
    role = Role;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 Amazon SageMaker.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policyConfig = {
```

```
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: [
          "sqs:ReceiveMessage",
          "sqs:DeleteMessage",
          "sqs:GetQueueAttributes",
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents",
          "sagemaker-geospatial:StartVectorEnrichmentJob",
          "sagemaker-geospatial:GetVectorEnrichmentJob",
          "sagemaker:SendPipelineExecutionStepFailure",
          "sagemaker:SendPipelineExecutionStepSuccess",
          "sagemaker-geospatial:ExportVectorEnrichmentJob",
        ],
        Resource: "*",
      },
      {
        Effect: "Allow",
        // The AWS Lambda function needs permission to pass the pipeline execution
 role to
        // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
 function
        // from elevating privileges. For more information, see:
        // https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_use_passrole.html
        Action: ["iam:PassRole"],
        Resource: `${pipelineExecutionRoleArn}`,
        Condition: {
          StringEquals: {
            "iam:PassedToService": [
              "sagemaker.amazonaws.com",
              "sagemaker-geospatial.amazonaws.com",
            ],
          },
        },
      },
    ],
  };

  const createPolicy = () =>
    iamClient.send(
```

```
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
        PolicyName: name,
      }),
    );

  let policy = null;

  try {
    const { Policy } = await createPolicy();
    policy = Policy;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}

/**
 * Attach an AWS IAM policy to an AWS IAM role.
 * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-
iam').IAMClient}} props
 */
export async function attachPolicy({ roleName, policyArn, iamClient }) {
  const attachPolicyCommand = new AttachRolePolicyCommand({
    RoleName: roleName,
```

```
    PolicyArn: policyArn,
  });

  await iamClient.send(attachPolicyCommand);
  return {
    cleanUp: async () => {
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: roleName,
          PolicyArn: policyArn,
        }),
      );
    },
  };
}

/**
 * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon
 SageMaker Geospatial clients
 * in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
 Amazon SageMaker
 * Geospatial client wasn't introduced until v3.221.0.
 * @param {{ name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient }} props
 */
export async function createLambdaLayer({ name, lambdaClient }) {
  const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;
  const { LayerVersionArn, Version } = await lambdaClient.send(
    new PublishLayerVersionCommand({
      LayerName: name,
      Content: {
        ZipFile: Uint8Array.from(readFileSync(layerPath)),
      },
    }),
  );

  return {
    versionArn: LayerVersionArn,
    version: Version,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteLayerVersionCommand({
          LayerName: name,
          VersionNumber: Version,
```

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

/**
 * Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
 pipeline
 * execution steps.
 * @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient, layerVersionArn: string}} props
 */
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
  layerVersionArn,
}) {
  const lambdaPath = `${dirnameFromMetaUrl(
    import.meta.url,
  )}lambda/dist/index.mjs.zip`;

  // If a function of the same name already exists, return that
  // function's ARN instead. By default this is
  // "sagemaker-wkflw-lambda-function", so collisions are
  // unlikely.
  const createFunction = async () => {
    try {
      return await lambdaClient.send(
        new CreateFunctionCommand({
          Code: {
            ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
          },
          Runtime: Runtime.nodejs18x,
          Handler: "index.handler",
          Layers: [layerVersionArn],
          FunctionName: name,
          Role: roleArn,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
```

```
        caught.name === "ResourceConflictException"
      ) {
        const { Configuration } = await lambdaClient.send(
          new GetFunctionCommand({ FunctionName: name }),
        );
        return Configuration;
      } else {
        throw caught;
      }
    }
  };

  // Function creation fails if the Role is not ready. This retries
  // function creation until it succeeds or it times out.
  const { FunctionArn } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    createFunction,
  );

  return {
    arn: FunctionArn,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteFunctionCommand({ FunctionName: name }),
      );
    },
  };
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
 props
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
  const s3Path = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../../workflows/sagemaker_pipelines/resources/latlongtest.csv`;

  await s3Client.send(
    new PutObjectCommand({
```

```
      Bucket: bucketName,
      Key: "input/sample_data.csv",
      Body: readFileSync(s3Path),
    }),
  );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient, wait:
 (ms: number) => Promise<void>}} props
 */
export async function createSagemakerRole({ name, iamClient, wait }) {
  let role = null;

  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: {
                Service: [
                  "sagemaker.amazonaws.com",
                  "sagemaker-geospatial.amazonaws.com",
                ],
              },
            },
          ],
        }),
      }),
    );

  try {
    const { Role } = await createRole();
    role = Role;
    // Wait for the role to be ready.
    await wait(10);
  } catch (caught) {
    if (
```

```
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 messages to
 * the Amazon SQS queue.
 * @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
 import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
 */
export async function createSagemakerExecutionPolicy({
  sqsQueueArn,
  lambdaArn,
  iamClient,
  name,
  s3BucketName,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: ["lambda:InvokeFunction"],
        Resource: lambdaArn,
      },
      {
```

```
      Effect: "Allow",
      Action: ["s3:*"],
      Resource: [
        `arn:aws:s3:::${s3BucketName}`,
        `arn:aws:s3:::${s3BucketName}/*`,
      ],
    },
    {
      Effect: "Allow",
      Action: ["sqs:SendMessage"],
      Resource: sqsQueueArn,
    },
  ],
};

const createPolicy = () =>
  iamClient.send(
    new CreatePolicyCommand({
      PolicyDocument: JSON.stringify(policyConfig),
      PolicyName: name,
    }),
  );

let policy = null;

try {
  const { Policy } = await createPolicy();
  policy = Policy;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
    if (Policies) {
      policy = Policies.find((p) => p.PolicyName === name);
    } else {
      throw new Error("No policies found.");
    }
  } else {
    throw caught;
  }
}
```

```
  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}

/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
```

```
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
  const createSqsQueue = () =>
    sqsClient.send(
      new CreateQueueCommand({
        QueueName: name,
        Attributes: {
```

```
          DelaySeconds: "5",
          ReceiveMessageWaitTimeSeconds: "5",
          VisibilityTimeout: "300",
        },
      }),
    );

  let queueUrl = null;
  try {
    const { QueueUrl } = await createSqsQueue();
    queueUrl = QueueUrl;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "QueueNameExists") {
      const { QueueUrl } = await sqsClient.send(
        new GetQueueUrlCommand({ QueueName: name }),
      );
      queueUrl = QueueUrl;
    } else {
      throw caught;
    }
  }

  const { Attributes } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    () =>
      sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: queueUrl,
          AttributeNames: ["QueueArn"],
        }),
      ),
  );

  return {
    queueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl: queueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
```

```
 * queue.
 * @param {{
 *   paginateListEventSourceMappings: () => Generator<import('@aws-sdk/client-
lambda').ListEventSourceMappingsCommandOutput>,
 *   lambdaName: string,
 *   queueArn: string,
 *   lambdaClient: import('@aws-sdk/client-lambda').LambdaClient}} props
 */
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
  paginateListEventSourceMappings,
}) {
  let uuid = null;
  const createEvenSourceMapping = () =>
    lambdaClient.send(
      new CreateEventSourceMappingCommand({
        EventSourceArn: queueArn,
        FunctionName: lambdaName,
      }),
    );

  try {
    const { UUID } = await createEvenSourceMapping();
    uuid = UUID;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceConflictException"
    ) {
      const paginator = paginateListEventSourceMappings(
        { client: lambdaClient },
        {},
      );
      /**
       * @type {import('@aws-sdk/client-lambda').EventSourceMappingConfiguration[]}
       */
      const eventSourceMappings = [];
      for await (const page of paginator) {
        eventSourceMappings.concat(page.EventSourceMappings || []);
      }

      const { Configuration } = await lambdaClient.send(
```

```
        new GetFunctionCommand({ FunctionName: lambdaName }),
      );

      uuid = eventSourceMappings.find(
        (mapping) =>
          mapping.EventSourceArn === queueArn &&
          mapping.FunctionArn === Configuration.FunctionArn,
      ).UUID;
    } else {
      throw caught;
    }
  }

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
          UUID: uuid,
        }),
      );
    },
  };
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{
 *   s3Client: import('@aws-sdk/client-s3').S3Client,
 *   name: string,
 *   paginateListObjectsV2: () => Generator<import('@aws-sdk/client-
s3').ListObjectsCommandOutput>
 * }} props
 */
export async function createS3Bucket({
  name,
  s3Client,
  paginateListObjectsV2,
}) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(
```

```
        { client: s3Client },
        { Bucket: name },
      );
      for await (const page of paginator) {
        const objects = page.Contents;
        if (objects) {
          for (const object of objects) {
            await s3Client.send(
              new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
            );
          }
        }
      }
      await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
    },
  };
}

/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
```

```
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
```

```
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}

/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
```

```
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-
s3').S3Client}} param0
 */
export async function getObject({ bucket, s3Client }) {
  const prefix = "output/";
  const { Contents } = await s3Client.send(
    new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
  );

  if (!Contents.length) {
    throw new Error("No objects found in bucket.");
  }

  // Find the CSV file.
  const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

  if (!outputObject) {
    throw new Error(`No CSV file found in bucket with the prefix "${prefix}".`);
  }

  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: outputObject.Key,
    }),
  );
```

```
  return Body.transformToString();
}
```

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 {
      this.logger.logSeparator();
      const doCleanUp = await this.prompter.confirm({
        message: "Clean up resources?",
      });
      if (doCleanUp) {
        await this.cleanUp();
      }
    }
  }

  async cleanUp() {
    // Run all of the clean up functions. If any fail, we log the error and
 continue.
    // This ensures all clean up functions are run.
    for (let i = this.cleanUpFunctions.length - 1; i >= 0; i--) {
      await retry(
        { intervalInMs: 1000, maxRetries: 60, swallowError: true },
        this.cleanUpFunctions[i],
      );
    }
  }
```

```
 async startWorkflow() {
   this.logger.logSeparator(MESSAGES.greetingHeader);
   await this.logger.log(MESSAGES.greeting);

   this.logger.logSeparator();
   await this.logger.log(
     MESSAGES.creatingRole.replace(
       "${ROLE_NAME}",
       this.names.LAMBDA_EXECUTION_ROLE,
     ),
   );

   // Create an IAM role that will be assumed by the AWS Lambda function. This
function
   // is triggered by Amazon SQS messages and calls SageMaker and SageMaker
GeoSpatial actions.
   const { arn: lambdaExecutionRoleArn, cleanUp: lambdaExecutionRoleCleanUp } =
     await createLambdaExecutionRole({
       name: this.names.LAMBDA_EXECUTION_ROLE,
       iamClient: this.clients.IAM,
     });
   // Add a clean up step to a stack for every resource created.
   this.cleanUpFunctions.push(lambdaExecutionRoleCleanUp);

   await this.logger.log(
     MESSAGES.roleCreated.replace(
       "${ROLE_NAME}",
       this.names.LAMBDA_EXECUTION_ROLE,
     ),
   );

   this.logger.logSeparator();

   await this.logger.log(
     MESSAGES.creatingRole.replace(
       "${ROLE_NAME}",
       this.names.SAGE_MAKER_EXECUTION_ROLE,
     ),
   );

   // Create an IAM role that will be assumed by the SageMaker pipeline. The
pipeline
   // sends messages to an Amazon SQS queue and puts/retrieves Amazon S3 objects.
   const {
```

```
      arn: pipelineExecutionRoleArn,
      cleanUp: pipelineExecutionRoleCleanUp,
    } = await createSagemakerRole({
      iamClient: this.clients.IAM,
      name: this.names.SAGE_MAKER_EXECUTION_ROLE,
      wait,
    });
    this.cleanUpFunctions.push(pipelineExecutionRoleCleanUp);

    await this.logger.log(
      MESSAGES.roleCreated.replace(
        "${ROLE_NAME}",
        this.names.SAGE_MAKER_EXECUTION_ROLE,
      ),
    );

    this.logger.logSeparator();

    // Create an IAM policy that allows the AWS Lambda function to invoke SageMaker
APIs.
    const {
      arn: lambdaExecutionPolicyArn,
      policy: lambdaPolicy,
      cleanUp: lambdaExecutionPolicyCleanUp,
    } = await createLambdaExecutionPolicy({
      name: this.names.LAMBDA_EXECUTION_ROLE_POLICY,
      s3BucketName: this.names.S3_BUCKET,
      iamClient: this.clients.IAM,
      pipelineExecutionRoleArn,
    });
    this.cleanUpFunctions.push(lambdaExecutionPolicyCleanUp);

    console.log(JSON.stringify(lambdaPolicy, null, 2), "\n");

    await this.logger.log(
      MESSAGES.attachPolicy
        .replace("${POLICY_NAME}", this.names.LAMBDA_EXECUTION_ROLE_POLICY)
        .replace("${ROLE_NAME}", this.names.LAMBDA_EXECUTION_ROLE),
    );

    await this.prompter.checkContinue();

    // Attach the Lambda execution policy to the execution role.
    const { cleanUp: lambdaExecutionRolePolicyCleanUp } = await attachPolicy({
```

```
    roleName: this.names.LAMBDA_EXECUTION_ROLE,
    policyArn: lambdaExecutionPolicyArn,
    iamClient: this.clients.IAM,
  });
  this.cleanUpFunctions.push(lambdaExecutionRolePolicyCleanUp);

  await this.logger.log(MESSAGES.policyAttached);

  this.logger.logSeparator();

  // Create Lambda layer for SageMaker packages.
  const { versionArn: layerVersionArn, cleanUp: lambdaLayerCleanUp } =
    await createLambdaLayer({
      name: this.names.LAMBDA_LAYER,
      lambdaClient: this.clients.Lambda,
    });
  this.cleanUpFunctions.push(lambdaLayerCleanUp);

  await this.logger.log(
    MESSAGES.creatingFunction.replace(
      "${FUNCTION_NAME}",
      this.names.LAMBDA_FUNCTION,
    ),
  );

  // Create the Lambda function with the execution role.
  const { arn: lambdaArn, cleanUp: lambdaCleanUp } =
    await createLambdaFunction({
      roleArn: lambdaExecutionRoleArn,
      lambdaClient: this.clients.Lambda,
      name: this.names.LAMBDA_FUNCTION,
      layerVersionArn,
    });
  this.cleanUpFunctions.push(lambdaCleanUp);

  await this.logger.log(
    MESSAGES.functionCreated.replace(
      "${FUNCTION_NAME}",
      this.names.LAMBDA_FUNCTION,
    ),
  );

  this.logger.logSeparator();
```

```
    await this.logger.log(
      MESSAGES.creatingSQSQueue.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
    );

    // Create an SQS queue for the SageMaker pipeline.
    const {
      queueUrl,
      queueArn,
      cleanUp: queueCleanUp,
    } = await createSQSQueue({
      name: this.names.SQS_QUEUE,
      sqsClient: this.clients.SQS,
    });
    this.cleanUpFunctions.push(queueCleanUp);

    await this.logger.log(
      MESSAGES.sqsQueueCreated.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.configuringLambdaSQSEventSource
        .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
        .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
    );

    // Configure the SQS queue as an event source for the Lambda.
    const { cleanUp: lambdaSQSEventSourceCleanUp } =
      await configureLambdaSQSEventSource({
        lambdaArn,
        lambdaName: this.names.LAMBDA_FUNCTION,
        queueArn,
        sqsClient: this.clients.SQS,
        lambdaClient: this.clients.Lambda,
      });
    this.cleanUpFunctions.push(lambdaSQSEventSourceCleanUp);

    await this.logger.log(
      MESSAGES.lambdaSQSEventSourceConfigured
        .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
        .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
    );
```

```
    this.logger.logSeparator();

    // Create an IAM policy that allows the SageMaker pipeline to invoke AWS Lambda
    // and send messages to the Amazon SQS queue.
    const {
      arn: pipelineExecutionPolicyArn,
      policy: sagemakerPolicy,
      cleanUp: pipelineExecutionPolicyCleanUp,
    } = await createSagemakerExecutionPolicy({
      sqsQueueArn: queueArn,
      lambdaArn,
      iamClient: this.clients.IAM,
      name: this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY,
      s3BucketName: this.names.S3_BUCKET,
    });
    this.cleanUpFunctions.push(pipelineExecutionPolicyCleanUp);

    console.log(JSON.stringify(sagemakerPolicy, null, 2));

    await this.logger.log(
      MESSAGES.attachPolicy
        .replace("${POLICY_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY)
        .replace("${ROLE_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE),
    );

    await this.prompter.checkContinue();

    // Attach the SageMaker execution policy to the execution role.
    const { cleanUp: pipelineExecutionRolePolicyCleanUp } = await attachPolicy({
      roleName: this.names.SAGE_MAKER_EXECUTION_ROLE,
      policyArn: pipelineExecutionPolicyArn,
      iamClient: this.clients.IAM,
    });
    this.cleanUpFunctions.push(pipelineExecutionRolePolicyCleanUp);
    // Wait for the role to be ready. If the role is used immediately,
    // the pipeline will fail.
    await wait(5);

    await this.logger.log(MESSAGES.policyAttached);

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingPipeline.replace(
```

```
        "${PIPELINE_NAME}",
        this.names.SAGE_MAKER_PIPELINE,
      ),
    );

    // Create the SageMaker pipeline.
    const { cleanUp: pipelineCleanUp } = await createSagemakerPipeline({
      roleArn: pipelineExecutionRoleArn,
      functionArn: lambdaArn,
      sagemakerClient: this.clients.SageMaker,
      name: this.names.SAGE_MAKER_PIPELINE,
    });
    this.cleanUpFunctions.push(pipelineCleanUp);

    await this.logger.log(
      MESSAGES.pipelineCreated.replace(
        "${PIPELINE_NAME}",
        this.names.SAGE_MAKER_PIPELINE,
      ),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingS3Bucket.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
    );

    // Create an S3 bucket for storing inputs and outputs.
    const { cleanUp: s3BucketCleanUp } = await createS3Bucket({
      name: this.names.S3_BUCKET,
      s3Client: this.clients.S3,
    });
    this.cleanUpFunctions.push(s3BucketCleanUp);

    await this.logger.log(
      MESSAGES.s3BucketCreated.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.uploadingInputData.replace(
        "${BUCKET_NAME}",
        this.names.S3_BUCKET,
```

```
    ),
  );

  // Upload CSV Lat/Long data to S3.
  await uploadCSVDataToS3({
    bucketName: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
  });

  await this.logger.log(MESSAGES.inputDataUploaded);

  this.logger.logSeparator();

  await this.prompter.checkContinue(MESSAGES.executePipeline);

  // Execute the SageMaker pipeline.
  const { arn: pipelineExecutionArn } = await startPipelineExecution({
    name: this.names.SAGE_MAKER_PIPELINE,
    sagemakerClient: this.clients.SageMaker,
    roleArn: pipelineExecutionRoleArn,
    bucketName: this.names.S3_BUCKET,
    queueUrl,
  });

  // Wait for the pipeline execution to finish.
  await waitForPipelineComplete({
    arn: pipelineExecutionArn,
    sagemakerClient: this.clients.SageMaker,
    wait,
  });

  this.logger.logSeparator();

  await this.logger.log(MESSAGES.outputDelay);

  // The getOutput function will throw an error if the output is not
  // found. The retry function will retry a failed function call once
  // ever 10 seconds for 2 minutes.
  const output = await retry({ intervalInMs: 10000, maxRetries: 12 }, () =>
    getObject({
      bucket: this.names.S3_BUCKET,
      s3Client: this.clients.S3,
    }),
  );
```

```
    this.logger.logSeparator();
    await this.logger.log(MESSAGES.outputDataRetrieved);
    console.log(output.split("\n").slice(0, 6).join("\n"));
  }
}
```

- Per API i dettagli, vedere i seguenti argomenti in Reference.AWS SDK for JavaScript API
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution
  - UpdatePipeline

Esempi di Secrets Manager 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 Secrets Manager.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`GetSecretValue`

Il seguente esempio di codice mostra come utilizzare`GetSecretValue`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 API i dettagli, vedi GetSecretValue AWS SDK for JavaScriptAPIReference.

SESEsempi di Amazon 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 AmazonSES.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni
- Scenari

Azioni

`CreateReceiptFilter`

Il seguente esempio di codice mostra come utilizzare`CreateReceiptFilter`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
 address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
 addressesOptions.
      },
      /*
        The name of the IP address filter. Only ASCII letters, numbers, underscores,
 or dashes.
        Must be less than 64 characters and start and end with a letter or number.
       */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });
```

```
  try {
    return await sesClient.send(createReceiptFilterCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Per API i dettagli, vedi CreateReceiptFilter AWS SDK for JavaScriptAPIReference.

`CreateReceiptRule`

Il seguente esempio di codice mostra come utilizzare`CreateReceiptRule`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi CreateReceiptRule AWS SDK for JavaScriptAPIReference.

`CreateReceiptRuleSet`

Il seguente esempio di codice mostra come utilizzare`CreateReceiptRuleSet`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi CreateReceiptRuleSet AWS SDK for JavaScriptAPIReference.

`CreateTemplate`

Il seguente esempio di codice mostra come utilizzare`CreateTemplate`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi CreateTemplate AWS SDK for JavaScriptAPIReference.

`DeleteIdentity`

Il seguente esempio di codice mostra come utilizzare`DeleteIdentity`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteIdentity AWS SDK for JavaScriptAPIReference.

`DeleteReceiptFilter`

Il seguente esempio di codice mostra come utilizzare`DeleteReceiptFilter`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteReceiptFilter AWS SDK for JavaScriptAPIReference.

`DeleteReceiptRule`

Il seguente esempio di codice mostra come utilizzare`DeleteReceiptRule`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteReceiptRule AWS SDK for JavaScriptAPIReference.

`DeleteReceiptRuleSet`

Il seguente esempio di codice mostra come utilizzare`DeleteReceiptRuleSet`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteReceiptRuleSet AWS SDK for JavaScriptAPIReference.

`DeleteTemplate`

Il seguente esempio di codice mostra come utilizzare`DeleteTemplate`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DeleteTemplate AWS SDK for JavaScriptAPIReference.

`GetTemplate`

Il seguente esempio di codice mostra come utilizzare`GetTemplate`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

const TEMPLATE_NAME = getUniqueName("TemplateName");

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

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

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Per API i dettagli, vedi GetTemplate AWS SDK for JavaScriptAPIReference.

`ListIdentities`

Il seguente esempio di codice mostra come utilizzare`ListIdentities`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListIdentities AWS SDK for JavaScriptAPIReference.

`ListReceiptFilters`

Il seguente esempio di codice mostra come utilizzare`ListReceiptFilters`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListReceiptFilters AWS SDK for JavaScriptAPIReference.

`ListTemplates`

Il seguente esempio di codice mostra come utilizzare`ListTemplates`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ListTemplates AWS SDK for JavaScriptAPIReference.

`SendBulkTemplatedEmail`

Il seguente esempio di codice mostra come utilizzare`SendBulkTemplatedEmail`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

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

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

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

/**
 *
```

```
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map
 each user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
```

```
};
```

- Per API i dettagli, vedi SendBulkTemplatedEmail AWS SDK for JavaScriptAPIReference.

`SendEmail`

Il seguente esempio di codice mostra come utilizzare`SendEmail`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

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

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

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

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Per API i dettagli, vedi SendEmail AWS SDK for JavaScriptAPIReference.

`SendRawEmail`

Il seguente esempio di codice mostra come utilizzare`SendRawEmail`.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi SendRawEmail AWS SDK for JavaScriptAPIReference.

`SendTemplatedEmail`

Il seguente esempio di codice mostra come utilizzare`SendTemplatedEmail`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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

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

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

```
const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
 party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Per API i dettagli, vedi SendTemplatedEmail AWS SDK for JavaScriptAPIReference.

`UpdateTemplate`

Il seguente esempio di codice mostra come utilizzare`UpdateTemplate`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi UpdateTemplate AWS SDK for JavaScriptAPIReference.

`VerifyDomainIdentity`

Il seguente esempio di codice mostra come utilizzare`VerifyDomainIdentity`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi VerifyDomainIdentity AWS SDK for JavaScriptAPIReference.

`VerifyEmailIdentity`

Il seguente esempio di codice mostra come utilizzare`VerifyEmailIdentity`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Import 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 API i dettagli, vedi VerifyEmailIdentity AWS SDK for JavaScriptAPIReference.

Scenari

Creazione di un'app in streaming Amazon Transcribe

L'esempio di codice seguente mostra come creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia tramite e-mail i risultati.

SDKper JavaScript (v3)

Mostra come usare Amazon Transcribe per creare un'app che registra, trascrive e traduce audio dal vivo in tempo reale e invia tramite e-mail i risultati utilizzando Amazon Simple Email Service (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 Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Creazione di un'applicazione Web per tracciare i dati DynamoDB

Il seguente esempio di codice mostra come creare un'applicazione Web che tiene traccia degli elementi di lavoro in una tabella Amazon DynamoDB e utilizza Amazon Simple Email Service (SESAmazon) per inviare report.

SDKper JavaScript (v3)

Mostra come utilizzare Amazon DynamoDB per creare un'applicazione Web API dinamica che tenga traccia dei dati di lavoro di DynamoDB.

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 SES

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

Il seguente esempio di codice mostra come creare un'applicazione Web che tiene traccia degli elementi di lavoro in un database Amazon Aurora Serverless e utilizza Amazon Simple Email Service (AmazonSES) per inviare report.

SDKper 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 (AmazonSES). 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. AWS servizi
- Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
- Invia un report via e-mail sugli elementi di lavoro filtrati utilizzando AmazonSES.
- Distribuisci e gestisci risorse di esempio con lo script incluso. AWS CloudFormation

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
- Servizio RDS dati Amazon
- Amazon SES

Rileva PPE nelle immagini

Il seguente esempio di codice mostra come creare un'app che utilizza Amazon Rekognition per rilevare i dispositivi di protezione PPE individuale () nelle immagini.

SDKper JavaScript (v3)

Mostra come usare Amazon Rekognition AWS SDK for JavaScript con per creare un'applicazione per rilevare dispositivi di protezione PPE individuale () nelle immagini che si trovano in un bucket Amazon Simple Storage Service (Amazon S3). L'app salva i risultati in una tabella Amazon

DynamoDB e invia all'amministratore una notifica e-mail con i risultati utilizzando Amazon Simple Email Service (Amazon). SES

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per PPE usare Amazon Rekognition.
- Verifica un indirizzo email per AmazonSES.
- Aggiornare una tabella DynamoDB con i risultati.
- Invia una notifica via e-mail tramite AmazonSES.

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 Rekognition
- Amazon S3
- Amazon SES

Rilevamento di oggetti nelle immagini

Il seguente esempio di codice mostra come creare un'app che utilizza Amazon Rekognition per rilevare oggetti per categoria nelle immagini.

SDKper JavaScript (v3)

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'app invia all'amministratore una notifica e-mail con i risultati utilizzando Amazon Simple Email Service (AmazonSES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per rilevare gli oggetti tramite Amazon Rekognition.
- Verifica un indirizzo email per AmazonSES.
- Invia una notifica via e-mail tramite AmazonSES.

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 Rekognition
- Amazon S3
- Amazon SES

SNSEsempi di Amazon 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 AmazonSNS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Ciao Amazon SNS

I seguenti esempi di codice mostrano come iniziare a usare AmazonSNS.

SDKper JavaScript (v3)

> **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.

Inizializza un SNS client 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 API i dettagli, vedi ListTopics AWS SDK for JavaScriptAPIReference.

Argomenti

- Azioni
- Scenari
- Esempi serverless

Azioni

`CheckIfPhoneNumberIsOptedOut`

Il seguente esempio di codice mostra come utilizzare`CheckIfPhoneNumberIsOptedOut`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere CheckIfPhoneNumberIsOptedOutin AWS SDK for JavaScript APIReference.

`ConfirmSubscription`

Il seguente esempio di codice mostra come utilizzare`ConfirmSubscription`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere ConfirmSubscriptionin AWS SDK for JavaScript APIReference.

`CreateTopic`

Il seguente esempio di codice mostra come utilizzare`CreateTopic`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere CreateTopicin AWS SDK for JavaScript APIReference.

`DeleteTopic`

Il seguente esempio di codice mostra come utilizzare`DeleteTopic`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere DeleteTopicin AWS SDK for JavaScript APIReference.

`GetSMSAttributes`

Il seguente esempio di codice mostra come utilizzare`GetSMSAttributes`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere G etSMSAttributes in AWS SDK for JavaScript APIReference.

`GetTopicAttributes`

Il seguente esempio di codice mostra come utilizzare`GetTopicAttributes`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

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

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1: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 API i dettagli, vedere GetTopicAttributesin AWS SDK for JavaScript APIReference.

SDKper 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.

Importa i moduli SDK e client e chiama ilAPI.

```
// 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 API i dettagli, vedere GetTopicAttributesin AWS SDK for JavaScript APIReference.

`ListSubscriptions`

Il seguente esempio di codice mostra come utilizzare`ListSubscriptions`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere ListSubscriptionsin AWS SDK for JavaScript APIReference.

`ListTopics`

Il seguente esempio di codice mostra come utilizzare`ListTopics`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere ListTopicsin AWS SDK for JavaScript APIReference.

`Publish`

Il seguente esempio di codice mostra come utilizzare`Publish`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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;
};
```

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 ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per API i dettagli, consulta Publish in AWS SDK for JavaScript APIReference.

`SetSMSAttributes`

Il seguente esempio di codice mostra come utilizzare`SetSMSAttributes`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere S etSMSAttributes in AWS SDK for JavaScript APIReference.

`SetTopicAttributes`

Il seguente esempio di codice mostra come utilizzare`SetTopicAttributes`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, vedere SetTopicAttributesin AWS SDK for JavaScript APIReference.

`Subscribe`

Il seguente esempio di codice mostra come utilizzare`Subscribe`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

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

Sottoscrivi un'applicazione mobile a un 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;
};
```

Sottoscrivi una funzione Lambda a un 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 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;
};
```

Sottoscrivi una SQS coda a un argomento.

```
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;
};
```

Sottoscrivi con un filtro a un argomento.

```
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 ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per API i dettagli, consulta Subscribe in AWS SDK for JavaScript APIReference.

`Unsubscribe`

Il seguente esempio di codice mostra come utilizzare`Unsubscribe`.

SDKper JavaScript (v3)

> **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.

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({});
```

Importa i moduli SDK e client e chiama ilAPI.

```
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 API i dettagli, consulta Annullare l'iscrizione in AWS SDK for JavaScript APIReference.

Scenari

Costruisci un'app per inviare dati a una tabella DynamoDB

Il seguente esempio di codice mostra come creare un'applicazione che invia dati a una tabella Amazon DynamoDB e ti avvisa quando un utente aggiorna la tabella.

SDKper (v3) JavaScript

Questo esempio mostra come creare un'app che consenta agli utenti di inviare dati a una tabella Amazon DynamoDB e inviare 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, consulta 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

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDKper 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, vedi l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- APIGateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Creazione di un'applicazione Amazon Textract explorer

Il seguente esempio di codice mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDKper 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 Amazon Simple Queue Service (Amazon) sottoscritta a un argomento SQS Amazon Simple Notification Service (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

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Crea argomento (FIFOo nonFIFO).
- 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.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Pubblicare
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Esempi serverless

Richiama una funzione Lambda da un trigger Amazon SNS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da un argomento. SNS La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumare un SNS evento utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumare un SNS evento utilizzando Lambda. TypeScript

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

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
```

```
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

SQSEsempi di Amazon 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 AmazonSQS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Ciao Amazon SQS

I seguenti esempi di codice mostrano come iniziare a usare AmazonSQS.

SDKper JavaScript (v3)

> **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.

Inizializza un SQS client Amazon 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 API i dettagli, consulta Reference ListQueues.AWS SDK for JavaScript API

Argomenti

- Azioni
- Scenari
- Esempi serverless

Azioni

`ChangeMessageVisibility`

Il seguente esempio di codice mostra come utilizzare`ChangeMessageVisibility`.

SDKper JavaScript (v3)

> **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.

Ricevi un SQS messaggio Amazon 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 API i dettagli, consulta ChangeMessageVisibility AWS SDK for JavaScriptAPIReference.

SDKper 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 SQS messaggio Amazon 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 API i dettagli, consulta ChangeMessageVisibility AWS SDK for JavaScriptAPIReference.

`CreateQueue`

Il seguente esempio di codice mostra come utilizzare`CreateQueue`.

SDKper JavaScript (v3)

> **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.

Crea una coda Amazon SQS standard.

```
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 SQS coda Amazon con sondaggi lunghi.

```
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 API i dettagli, consulta la sezione CreateQueueReference AWS SDK for JavaScript API.

SDKper 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 Amazon SQS standard.

```
// 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 SQS coda Amazon 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 API i dettagli, consulta Reference CreateQueue.AWS SDK for JavaScript API

`DeleteMessage`

Il seguente esempio di codice mostra come utilizzare`DeleteMessage`.

SDKper JavaScript (v3)

> **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.

Ricevi ed elimina SQS messaggi Amazon.

```
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 API i dettagli, DeleteMessageconsulta AWS SDK for JavaScript APIReference.

SDKper 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 SQS messaggi Amazon.

```
// 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 API i dettagli, DeleteMessageconsulta AWS SDK for JavaScript APIReference.

`DeleteMessageBatch`

Il seguente esempio di codice mostra come utilizzare`DeleteMessageBatch`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  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 API i dettagli, vedi DeleteMessageBatch AWS SDK for JavaScriptAPIReference.

`DeleteQueue`

Il seguente esempio di codice mostra come utilizzare`DeleteQueue`.

SDKper JavaScript (v3)

> **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.

Elimina una SQS coda Amazon.

```
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 API i dettagli, consulta DeleteQueue AWS SDK for JavaScriptAPIReference.

SDKper 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 una SQS coda Amazon.

```
// 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 API i dettagli, consulta DeleteQueue AWS SDK for JavaScriptAPIReference.

`GetQueueAttributes`

Il seguente esempio di codice mostra come utilizzare`GetQueueAttributes`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi GetQueueAttributes AWS SDK for JavaScriptAPIReference.

`GetQueueUrl`

Il seguente esempio di codice mostra come utilizzare`GetQueueUrl`.

SDKper JavaScript (v3)

> **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.

Scarica il file URL per una SQS coda Amazon.

```
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 API i dettagli, consulta la sezione GetQueueUrl AWS SDK for JavaScriptAPIReference.

SDKper 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.

Scarica il file URL per una SQS coda Amazon.

```
// 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 API i dettagli, consulta la sezione GetQueueUrl AWS SDK for JavaScriptAPIReference.

`ListQueues`

Il seguente esempio di codice mostra come utilizzare`ListQueues`.

SDKper JavaScript (v3)

> **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 le tue SQS code Amazon.

```
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 API i dettagli, consulta la sezione ListQueues AWS SDK for JavaScriptAPIReference.

SDKper 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 SQS code Amazon.

```
// 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 API i dettagli, consulta la sezione ListQueues AWS SDK for JavaScriptAPIReference.

`ReceiveMessage`

Il seguente esempio di codice mostra come utilizzare`ReceiveMessage`.

SDKper JavaScript (v3)

> **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.

Ricevi un messaggio da una SQS coda Amazon.

```
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 SQS coda Amazon 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 API dettagli, consulta ReceiveMessagela sezione Reference.AWS SDK for JavaScript API

SDKper 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 SQS coda Amazon 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 API dettagli, consulta ReceiveMessagela sezione Reference.AWS SDK for JavaScript API

`SendMessage`

Il seguente esempio di codice mostra come utilizzare`SendMessage`.

SDKper JavaScript (v3)

> **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.

Invia un messaggio a una SQS coda Amazon.

```
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 API i dettagli, consulta la sezione SendMessage AWS SDK for JavaScriptAPIReference.

SDKper 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 SQS coda Amazon.

```
// 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 API i dettagli, consulta la sezione SendMessage AWS SDK for JavaScriptAPIReference.

`SetQueueAttributes`

Il seguente esempio di codice mostra come utilizzare`SetQueueAttributes`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 SQS coda Amazon 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;
};
```

Configura una coda dead-letter.

```
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 API i dettagli, consulta Reference [SetQueueAttributes](#).AWS SDK for JavaScript API

Scenari

Creazione di un'applicazione Amazon Textract explorer

Il seguente esempio di codice mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDKper 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 Amazon Simple Queue Service (Amazon) sottoscritta a un argomento SQS Amazon Simple Notification Service (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

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Crea argomento (FIFOo nonFIFO).
- 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.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Pubblicare
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Esempi serverless

Richiama una funzione Lambda da un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda. SQS La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumare un SQS evento utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumare un SQS evento utilizzando Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Segnalazione degli errori degli articoli in batch per le funzioni Lambda con un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da una SQS coda. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi del SQS batch con JavaScript Lambda utilizzando.

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

Segnalazione degli errori degli elementi del SQS batch con TypeScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
```

```
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`StartExecution`

Il seguente esempio di codice mostra come utilizzare`StartExecution`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi StartExecution AWS SDK for JavaScriptAPIReference.

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 azioni mostrino come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`AssumeRole`

Il seguente esempio di codice mostra come utilizzare`AssumeRole`.

SDKper JavaScript (v3)

> **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.

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 IAM ruolo.

```
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 API i dettagli, vedere AssumeRolein AWS SDK for JavaScript APIReference.

SDKper 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 API i dettagli, vedi AssumeRole AWS SDK for JavaScriptAPIReference.

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

Gli elementi di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Mentre le azioni mostrano come richiamare le singole funzioni di servizio, è possibile visualizzare le azioni nel loro contesto nei relativi scenari.

Ogni esempio include un collegamento al codice sorgente completo, in cui 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.

SDKper JavaScript (v3)

> **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.

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 API i dettagli, vedi DescribeServices AWS SDK for JavaScriptAPIReference.

Argomenti

- Nozioni di base
- Azioni

Nozioni di base

Scopri le nozioni di base

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.

SDKper JavaScript (v3)

> **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.

Esegui uno scenario interattivo nel terminale.

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import * as inquirer from "@inquirer/prompts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};

const client = new SupportClient({ region: "us-east-1" });

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});

  try {
    await client.send(command);
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
```

```
    } else {
      throw err;
    }
  }
};

/**
 * Select a service from the list returned from DescribeServices.
 */
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const selectedService = await inquirer.select({
    message:
      "Select a service. Your support case will be created for this service. The
 list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
};

/**
 * @param {{ categories: import('@aws-sdk/client-support').Category[]}} service
 */
export const getCategory = async (service) => {
  const selectedCategory = await inquirer.select({
    message: "Select a category.",
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const selectedSeverityLevel = await inquirer.select({
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

/**
 * Create a new support case
```

```
 * @param {{
 *  selectedService: import('@aws-sdk/client-support').Service
 *  selectedCategory: import('@aws-sdk/client-support').Category
 *  selectedSeverityLevel: import('@aws-sdk/client-support').SeverityLevel
 * }} selections
 * @returns
 */
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodaysOpenCases = async () => {
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });

  const { cases } = await client.send(command);

  if (cases.length === 0) {
    throw new Error(
      "Unexpected number of cases. Expected more than 0 open cases.",
    );
  }
  return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
```

```
  const command = new AddAttachmentsToSetCommand({
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
  });
  const { communications } = await client.send(command);
  return communications;
};

/**
 * @param {import('@aws-sdk/client-support').Communication[]} communications
 */
export const getFirstAttachment = (communications) => {
  const firstCommWithAttachment = communications.find(
    (c) => c.attachmentSet.length > 0,
  );
  return firstCommWithAttachment?.attachmentSet[0].attachmentId;
};

// Get an attachment.
export const getAttachment = async (attachmentId) => {
  const command = new DescribeAttachmentCommand({
    attachmentId,
```

```
  });
  const { attachment } = await client.send(command);
  return attachment;
};

// Resolve the case matching the given case ID.
export const resolveCase = async (caseId) => {
  const shouldResolve = await inquirer.confirm({
    message: `Do you want to resolve ${caseId}?`,
  });

  if (shouldResolve) {
    const command = new ResolveCaseCommand({
      caseId: caseId,
    });

    await client.send(command);
    return true;
  }
  return false;
};

/**
 * Find a specific case in the list of provided cases by case ID.
 * If the case is not found, and the results are paginated, continue
 * paging through the results.
 * @param {{
 *   caseId: string,
 *   cases: import('@aws-sdk/client-support').CaseDetails[]
 *   nextToken: string
 * }} options
 * @returns
 */
export const findCase = async ({ caseId, cases, nextToken }) => {
  const foundCase = cases.find((c) => c.caseId === caseId);

  if (foundCase) {
    return foundCase;
  }

  if (nextToken) {
    const response = await client.send(
      new DescribeCasesCommand({
        nextToken,
```

```
        includeResolvedCases: true,
      }),
    );
    return findCase({
      caseId,
      cases: response.cases,
      nextToken: response.nextToken,
    });
  }

  throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
 select one.
    const selectedCategory = await getCategory(selectedService);

    // Provide the severity available severity levels for the account and prompt the
 user to select one.
```

```
  const selectedSeverityLevel = await getSeverityLevel();

  // Create a support case.
  console.log("\nCreating a support case.");
  caseId = await createCase({
    selectedService,
    selectedCategory,
    selectedSeverityLevel,
  });
  console.log(`Support case created: ${caseId}`);

  // Display a list of open support cases created today.
  const todaysOpenCases = await retry(
    { intervalInMs: 1000, maxRetries: 15 },
    getTodaysOpenCases,
  );
  console.log(
    `\nOpen support cases created today: ${todaysOpenCases.length}`,
  );
  console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

  // Create an attachment set.
  console.log("\nCreating an attachment set.");
  const attachmentSetId = await createAttachmentSet();
  console.log(`Attachment set created: ${attachmentSetId}`);

  // Add the attachment set to the support case.
  console.log(`\nAdding attachment set to ${caseId}`);
  await linkAttachmentSetToCase(attachmentSetId, caseId);
  console.log(`Attachment set added to ${caseId}`);

  // List the communications for a support case.
  console.log(`\nListing communications for ${caseId}`);
  const communications = await getCommunications(caseId);
  console.log(
    communications
      .map(
        (c) =>
          `Communication created on ${c.timeCreated}. Has
${c.attachmentSet.length} attachments.`,
      )
      .join("\n"),
  );
```

```
    // Describe the first attachment.
    console.log(`\nDescribing attachment ${attachmentSetId}`);
    const attachmentId = getFirstAttachment(communications);
    const attachment = await getAttachment(attachmentId);
    console.log(
      `Attachment is the file '${
        attachment.fileName
      }' with data: \n${new TextDecoder().decode(attachment.data)}`,
    );

    // Confirm that the support case should be resolved.
    const isResolved = await resolveCase(caseId);
    if (isResolved) {
      // List the resolved cases and include the one previously created.
      // Resolved cases can take a while to appear.
      console.log(
        "\nWaiting for case status to be marked as resolved. This can take some
 time.",
      );
      const resolvedCases = await retry(
        { intervalInMs: 20000, maxRetries: 15 },
        () => getTodaysResolvedCases(caseId),
      );
      console.log("Resolved cases:");
      console.log(resolvedCases.map((c) => c.caseId).join("\n"));
    }
  } catch (err) {
    console.error(err);
  }
};
```

- Per API i dettagli, consulta i seguenti argomenti in AWS SDK for JavaScript APIRiferimento.
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases
  - DescribeCommunications
  - DescribeServices

- DescribeSeverityLevels
- ResolveCase

Azioni

`AddAttachmentsToSet`

Il seguente esempio di codice mostra come utilizzare`AddAttachmentsToSet`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi AddAttachmentsToSet AWS SDK for JavaScriptAPIReference.

`AddCommunicationToCase`

Il seguente esempio di codice mostra come utilizzare`AddCommunicationToCase`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi AddCommunicationToCase AWS SDK for JavaScriptAPIReference.

`CreateCase`

Il seguente esempio di codice mostra come utilizzare`CreateCase`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi CreateCase AWS SDK for JavaScriptAPIReference.

`DescribeAttachment`

Il seguente esempio di codice mostra come utilizzare`DescribeAttachment`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DescribeAttachment AWS SDK for JavaScriptAPIReference.

`DescribeCases`

Il seguente esempio di codice mostra come utilizzare`DescribeCases`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DescribeCases AWS SDK for JavaScriptAPIReference.

`DescribeCommunications`

Il seguente esempio di codice mostra come utilizzare`DescribeCommunications`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DescribeCommunications AWS SDK for JavaScriptAPIReference.

`DescribeSeverityLevels`

Il seguente esempio di codice mostra come utilizzare`DescribeSeverityLevels`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi DescribeSeverityLevels AWS SDK for JavaScriptAPIReference.

`ResolveCase`

Il seguente esempio di codice mostra come utilizzare`ResolveCase`.

SDKper JavaScript (v3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 API i dettagli, vedi ResolveCase AWS SDK for JavaScriptAPIReference.

Esempi di Amazon Textract 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 Textract.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un'applicazione Amazon Textract explorer

Il seguente esempio di codice mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDKper 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 Amazon Simple Queue Service (Amazon) sottoscritta a un argomento SQS Amazon Simple Notification Service (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 per analizzare il feedback dei clienti

L'esempio di codice seguente mostra come creare un'applicazione che analizza le schede dei commenti dei clienti, le traduce dalla loro lingua originale, ne determina il sentiment e genera un file audio dal testo tradotto.

SDKper 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

Esempi di Amazon Transcribe SDK con 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. Mentre le azioni mostrano come chiamare le singole funzioni di servizio, puoi vedere le azioni nel loro contesto negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre AWS servizi.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`DeleteMedicalTranscriptionJob`

Il seguente esempio di codice mostra come utilizzare`DeleteMedicalTranscriptionJob`.

SDKper JavaScript (v3)

> **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.

Crea il client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
```

```
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

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 API i dettagli, vedi DeleteMedicalTranscriptionJob AWS SDK for JavaScriptAPIReference.

`DeleteTranscriptionJob`

Il seguente esempio di codice mostra come utilizzare`DeleteTranscriptionJob`.

SDKper JavaScript (v3)

> **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.

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.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per API i dettagli, vedi DeleteTranscriptionJob AWS SDK for JavaScriptAPIReference.

`ListMedicalTranscriptionJobs`

Il seguente esempio di codice mostra come utilizzare`ListMedicalTranscriptionJobs`.

SDKper JavaScript (v3)

> **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.

Crea il client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

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 API i dettagli, vedi ListMedicalTranscriptionJobs AWS SDK for JavaScriptAPIReference.

`ListTranscriptionJobs`

Il seguente esempio di codice mostra come utilizzare`ListTranscriptionJobs`.

SDKper JavaScript (v3)

> **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.

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.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per API i dettagli, vedi ListTranscriptionJobs AWS SDK for JavaScriptAPIReference.

`StartMedicalTranscriptionJob`

Il seguente esempio di codice mostra come utilizzare`StartMedicalTranscriptionJob`.

SDKper JavaScript (v3)

> **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.

Crea il client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

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 API i dettagli, vedi StartMedicalTranscriptionJob AWS SDK for JavaScriptAPIReference.

`StartTranscriptionJob`

Il seguente esempio di codice mostra come utilizzare`StartTranscriptionJob`.

SDKper JavaScript (v3)

> **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.

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.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per API i dettagli, vedi StartTranscriptionJob AWS SDK for JavaScriptAPIReference.

Scenari

Creazione di un'app in streaming Amazon Transcribe

L'esempio di codice seguente mostra come creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia tramite e-mail i risultati.

SDKper JavaScript (v3)

Mostra come usare Amazon Transcribe per creare un'app che registra, trascrive e traduce audio dal vivo in tempo reale e invia tramite e-mail i risultati utilizzando Amazon Simple Email Service (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 Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Esempi di Amazon Translate 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 Translate.

Gli scenari sono esempi di codice che mostrano come eseguire attività specifiche richiamando più funzioni all'interno di un servizio o combinandole con altre. AWS servizi

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Scenari

Scenari

Creazione di un'app in streaming Amazon Transcribe

L'esempio di codice seguente mostra come creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia tramite e-mail i risultati.

SDKper JavaScript (v3)

Mostra come usare Amazon Transcribe per creare un'app che registra, trascrive e traduce audio dal vivo in tempo reale e invia tramite e-mail i risultati utilizzando Amazon Simple Email Service (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 Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Creazione di un chatbot Amazon Lex

Il seguente esempio di codice mostra come creare un chatbot per coinvolgere i visitatori del tuo sito web.

SDKper JavaScript (v3)

Mostra come utilizzare Amazon Lex API 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

Crea un'applicazione per analizzare il feedback dei clienti

L'esempio di codice seguente mostra come creare un'applicazione che analizza le schede dei commenti dei clienti, le traduce dalla loro lingua originale, ne determina il sentiment e genera un file audio dal testo tradotto.

SDKper 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

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. L'utente è inoltre responsabile della configurazione della

protezione e delle attività di gestione per i AWS servizi utilizzati. Per ulteriori informazioni sulla privacy dei dati, consulta la sezione Privacy dei dati FAQ. Per informazioni sulla protezione dei dati in Europa, consulta il Modello di responsabilitàAWS condivisa e GDPR il post sul blog sulla AWS sicurezza.

Ai fini della protezione dei dati, ti consigliamo di proteggere Account AWS le credenziali e di configurare i singoli utenti con AWS IAM Identity Center o 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.
- UsaSSL/TLSper comunicare con AWS le risorse. Richiediamo TLS 1.2 e consigliamo TLS 1.3.
- Configurazione API e registrazione delle attività degli utenti con AWS CloudTrail.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno AWS servizi.
- 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 FIPS 140-3 moduli crittografici convalidati per accedere AWS tramite un'interfaccia a riga di comando o unAPI, usa un endpoint. FIPS Per ulteriori informazioni sugli FIPS endpoint disponibili, vedere Federal Information Processing Standard () 140-3. FIPS

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 si utilizza questo AWS prodotto o servizio o altro AWS servizi utilizzando la console, API AWS CLI, o. AWS SDKs 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. Se fornisci un URL a un server esterno, ti consigliamo vivamente di non includere le informazioni sulle credenziali URL per convalidare la tua richiesta a quel server.

Identity and Access Management

AWS Identity and Access Management (IAM) è un dispositivo AWS servizio che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. IAMgli amministratori controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle autorizzazioni) a utilizzare le risorse. AWS IAMè un dispositivo AWS servizio 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 svolgi. AWS

Utente del servizio: se lo utilizzi AWS servizi 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 relativi AWS all'identità e all'accesso o consulta la guida per l'utente della funzionalità AWS servizio 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. È quindi necessario inviare richieste all'IAMamministratore per modificare le autorizzazioni degli utenti del servizio. Consulta le informazioni contenute in questa pagina per comprendere i concetti di base diIAM. Per ulteriori informazioni su come la tua azienda può utilizzare IAM AWS, consulta la guida per l'utente del AWS servizio dispositivo che stai utilizzando.

IAMamministratore: se sei un IAM amministratore, potresti voler conoscere i dettagli su come scrivere politiche per gestire l'accesso AWS. Per visualizzare esempi di policy AWS basate sull'identità che puoi utilizzareIAM, consulta la guida per l'utente del programma AWS servizio che stai utilizzando.

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. È necessario autenticarsi (accedere a AWS) come Utente root dell'account AWS, come IAM utente o assumendo un ruolo. IAM

È possibile accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAMIdentity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Quando accedi come identità federata, l'amministratore aveva precedentemente configurato la federazione delle identità utilizzando i ruoli. IAM Quando si accede AWS utilizzando la federazione, si assume 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 () per firmare crittograficamente le tue richieste utilizzando le tue credenziali. CLI 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 Firmare AWS API le richieste nella Guida per l'IAMutente.

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'AWS IAM Identity Center utente e Utilizzo dell'autenticazione a più fattori (MFA) AWS nella Guida per l'IAMutente.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte AWS servizi 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 utilizzale per eseguire le operazioni che solo l'utente root può eseguire. Per l'elenco completo delle attività che richiedono l'accesso come utente root, consulta Attività che richiedono le credenziali dell'utente root nella Guida per l'IAMutente.

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 AWS servizi 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 AWS servizi 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 informazioni su IAM Identity Center, vedi Cos'è IAM Identity Center? nella Guida AWS IAM Identity Center per l'utente.

IAM users and groups

Un IAMutente è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Laddove possibile, consigliamo di fare affidamento su credenziali temporanee anziché creare IAM utenti con credenziali a lungo termine come password e chiavi di accesso. Tuttavia, se hai casi d'uso specifici che richiedono credenziali a lungo termine con IAM gli utenti, ti consigliamo di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta Ruotare regolarmente le chiavi di accesso per i casi d'uso che richiedono credenziali a lungo termine nella Guida per l'utente. IAM

Un IAMgruppo 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, potresti avere un gruppo denominato IAMAdminse concedere a quel gruppo le autorizzazioni per IAM amministrare le risorse.

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 IAM utente (anziché un ruolo) nella Guida per l'IAMutente.

IAMruoli

Un IAMruolo è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un IAM utente, ma non è associato a una persona specifica. È possibile assumere temporaneamente un IAM ruolo in AWS Management Console cambiando ruolo. È possibile assumere un ruolo chiamando un' AWS APIoperazione AWS CLI or o utilizzando un'operazione

personalizzataURL. Per ulteriori informazioni sui metodi di utilizzo dei ruoli, vedere Utilizzo IAM dei ruoli nella Guida per l'IAMutente.

IAMi ruoli 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 informazioni sui ruoli per la federazione, vedere Creazione di un ruolo per un provider di identità di terze parti nella Guida per l'IAMutente. Se utilizzi IAM Identity Center, configuri un set di autorizzazioni. Per controllare a cosa possono accedere le identità dopo l'autenticazione, IAM Identity Center correla il set di autorizzazioni a un ruolo in. IAM Per informazioni sui set di autorizzazioni, consulta Set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
- Autorizzazioni IAM utente temporanee: un IAM utente o un ruolo può assumere il IAM ruolo di assumere temporaneamente autorizzazioni diverse per un'attività specifica.
- Accesso su più account: puoi utilizzare un IAM ruolo per consentire a qualcuno (un responsabile fidato) di un altro account di accedere alle risorse del tuo account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni AWS servizi, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per conoscere la differenza tra ruoli e politiche basate sulle risorse per l'accesso tra account diversi, consulta la sezione Accesso alle risorse su più account IAM nella Guida per l'utente. IAM
- Accesso tra servizi: alcuni AWS servizi utilizzano funzionalità in altri. AWS servizi Ad esempio, quando effettui una chiamata in un servizio, è normale che quel 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 IAM utente o un ruolo per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FASutilizza le autorizzazioni del principale che chiama an AWS servizio, in combinazione con la richiesta di effettuare richieste AWS servizio ai servizi downstream. FASle richieste vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri AWS servizi o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli FAS delle politiche relative alle richieste, consulta Forward access sessions.
  - Ruolo di servizio: un ruolo di servizio è un IAMruolo che un servizio assume per eseguire azioni per conto dell'utente. Un IAM amministratore può creare, modificare ed eliminare un ruolo di

servizio dall'internoIAM. Per ulteriori informazioni, vedere Creazione di un ruolo per delegare le autorizzazioni a un utente AWS servizio nella Guida per l'IAMutente.

- Ruolo collegato al servizio: un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. AWS servizio 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 IAM amministratore può visualizzare, ma non modificare le autorizzazioni per i ruoli collegati al servizio.

- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un IAM ruolo per gestire le credenziali temporanee per le applicazioni in esecuzione su un'EC2istanza e che effettuano AWS CLI o richiedono AWS API. Ciò è preferibile alla memorizzazione delle chiavi di accesso all'interno dell'EC2istanza. Per assegnare un AWS ruolo a un'EC2istanza e renderlo disponibile per tutte le sue applicazioni, create un profilo di istanza collegato all'istanza. Un profilo di istanza contiene il ruolo e consente ai programmi in esecuzione sull'EC2istanza di ottenere credenziali temporanee. Per ulteriori informazioni, consulta Usare un IAM ruolo per concedere le autorizzazioni alle applicazioni in esecuzione su EC2 istanze Amazon nella Guida per l'IAMutente.

Per sapere se utilizzare IAM ruoli o IAM utenti, consulta Quando creare un IAM ruolo (anziché un utente) nella Guida per l'IAMutente.

Gestione dell'accesso con policy

Puoi controllare l'accesso AWS creando policy e associandole 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 JSON documenti. Per ulteriori informazioni sulla struttura e il contenuto dei documenti relativi alle JSON politiche, vedere Panoramica delle JSON politiche nella Guida per l'IAMutente.

Gli amministratori possono utilizzare AWS JSON le politiche 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 IAM amministratore può creare IAM politiche. L'amministratore può quindi aggiungere le IAM politiche ai ruoli e gli utenti possono assumerli.

IAMle politiche definiscono le autorizzazioni per un'azione indipendentemente dal metodo utilizzato per eseguire l'operazione. Ad esempio, supponiamo di disporre di una policy che consente l'operazione `iam:GetRole`. Un utente con tale criterio può ottenere informazioni sul ruolo da AWS Management Console, da o da. AWS CLI AWS API

Policy basate su identità

I criteri basati sull'identità sono documenti relativi alle politiche di JSON autorizzazione che è possibile allegare a un'identità, ad esempio un IAM utente, un gruppo di utenti o un ruolo. Tali policy definiscono le azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una politica basata sull'identità, consulta Creazione di politiche nella Guida per l'utente. IAM IAM

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono integrate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli all'interno del tuo. Account AWS Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una politica gestita o una politica in linea, consulta Scelta tra politiche gestite e politiche in linea nella Guida per l'IAMutente.

Policy basate su risorse

Le politiche basate sulle risorse sono documenti di JSON policy allegati a una risorsa. Esempi di politiche basate sulle risorse sono le policy di trust dei IAM ruoli e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata 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. AWS servizi

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le politiche AWS gestite IAM in una politica basata sulle risorse.

Elenchi di controllo degli accessi () ACLs

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLssono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy. JSON

Amazon S3 e Amazon VPC sono esempi di servizi che supportano. AWS WAF ACLs Per ulteriori informazioniACLs, consulta la panoramica di Access control list (ACL) nella Amazon Simple Storage Service Developer Guide.

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 di autorizzazioni è una funzionalità avanzata in cui si impostano le autorizzazioni massime che una politica basata sull'identità può concedere a un'entità (utente o ruolo). IAM 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 nella Guida per l'utente. IAM IAM
- Politiche di controllo del servizio (SCPs): SCPs sono JSON politiche 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 di un'organizzazione, puoi applicare le politiche di controllo del servizio (SCPs) a uno o tutti i tuoi account. SCPLimita le autorizzazioni per le entità negli account dei membri, inclusa ciascuna Utente root dell'account AWS. Per ulteriori informazioni su Organizations andSCPs, consulta le politiche di controllo dei servizi nella Guida AWS Organizations per l'utente.
- 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 le politiche di sessione nella Guida IAM per l'utente.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per informazioni su come AWS determinare se consentire una richiesta quando

sono coinvolti più tipi di policy, consulta Logica di valutazione delle politiche nella Guida per l'IAMutente.

Come AWS servizi lavorare con IAM

Per avere una panoramica generale del AWS servizi funzionamento della maggior parte delle IAM funzionalità, consulta AWS i servizi con cui è possibile utilizzare IAM nella Guida per l'IAMutente.

Per informazioni su come utilizzare una larghezza specifica AWS servizio IAM, consulta la sezione sulla sicurezza della Guida per l'utente del servizio pertinente.

Risoluzione dei problemi relativi AWS all'identità e all'accesso

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con AWS eIAM.

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 disponi dell'autorizzazione per eseguire un'operazione, le tue policy devono essere aggiornate in modo che ti sei consentito eseguire tale operazione.

L'errore di esempio seguente si verifica quando l'`mateojackson`IAMutente tenta di utilizzare la console per visualizzare i dettagli su una *`my-example-widget`* risorsa fittizia ma non dispone delle autorizzazioni fittizie`awes:`*`GetWidget`*.

```
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 assistenza, contatta l'amministratore. AWS L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un errore che indica che non sei autorizzato a eseguire l'operazione `iam:PassRole`, le tue policy devono essere aggiornate per poter passare un ruolo a AWS.

Alcuni AWS servizi 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 IAM utente denominato `marymajor` tenta di utilizzare la console per eseguire un'azione in. AWS Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per passare il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di assistenza, contatta AWS l'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 i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per concedere alle persone l'accesso alle tue risorse.

Per ulteriori informazioni, consulta gli argomenti seguenti:

- Per sapere se AWS supporta queste funzionalità, consulta. Come AWS servizi lavorare con IAM
- Per informazioni su Account AWS come fornire l'accesso alle risorse di tua proprietà, consulta Fornire l'accesso a un IAM utente di un altro Account AWS utente di tua proprietà nella Guida per l'IAMutente.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta Fornire l'accesso a persone Account AWS di proprietà di terzi nella Guida per l'IAMutente.

- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso agli utenti autenticati esternamente (federazione delle identità) nella Guida per l'IAMutente.
- Per conoscere la differenza tra l'utilizzo di ruoli e politiche basate sulle risorse per l'accesso tra account diversi, consulta la sezione Accesso alle risorse tra account nella Guida per l'utente. IAM IAM

Convalida della conformità per questo prodotto o servizio AWS

Per sapere se un AWS servizio programma rientra nell'ambito di specifici programmi di conformità, consulta AWS servizi la sezione Scope by Compliance Program AWS servizi e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di AWS conformità Programmi di 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 AWS servizi 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.
- Architettura per la HIPAA sicurezza e la conformità su Amazon Web Services: questo white paper descrive in che modo le aziende possono utilizzare AWS per creare applicazioni idonee. HIPAA

> **Note**
>
> Non tutte sono idonee. AWS servizi HIPAA Per ulteriori informazioni, consulta la Guida ai servizi HIPAA idonei.

- AWS Risorse per AWS per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe riguardare 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 AWS servizi 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

- Evaluating Resources with Rules nella Guida per gli AWS Config sviluppatori: il AWS Config servizio valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida del settore e alle normative.
- AWS Security Hub— Ciò AWS servizio 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.
- Amazon GuardDuty: AWS servizio rileva potenziali minacce ai tuoi carichi di lavoro Account AWS, ai contenitori e ai dati monitorando l'ambiente alla ricerca di attività sospette e dannose. GuardDuty può aiutarti a soddisfare vari requisiti di conformità, ad esempio PCI DSS soddisfacendo i requisiti di rilevamento delle intrusioni imposti da determinati framework di conformità.
- AWS Audit Manager— Ciò AWS servizio consente di verificare continuamente AWS l'utilizzo per semplificare la gestione del rischio 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.

Si utilizzano API chiamate AWS pubblicate per accedere a questo AWS Prodotto o Servizio attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). Richiediamo TLS 1.2 e consigliamo TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS) come (Ephemeral Diffie-Hellman) o DHE (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 minima TLS

Per aumentare la sicurezza durante la comunicazione con AWS i servizi, configurali AWS SDK for JavaScript per l'utilizzo della versione TLS 1.2 o successiva.

> **Important**
>
> La versione AWS SDK for JavaScript v3 negozia automaticamente la TLS versione di livello più alto supportata da un determinato AWS endpoint del servizio. Facoltativamente, puoi imporre una TLS versione minima richiesta dall'applicazione, ad esempio TLS 1.2 o 1.3, ma tieni presente che la 1.3 non è supportata da alcuni endpoint del AWS servizio, quindi alcune chiamate potrebbero non riuscire se applichi la TLS 1.3. TLS

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.

Verifica e applica in Node.js TLS

Quando si utilizza il AWS SDK for JavaScript con Node.js, il livello di sicurezza Node.js sottostante viene utilizzato per impostare la TLS versione.

Node.js 12.0.0 e versioni successive utilizzano una versione minima di Open SSL 1.1.1b, che supporta la versione 1.3. TLS Per impostazione predefinita, la AWS SDK for JavaScript v3 utilizza la TLS versione 1.3 quando disponibile, ma utilizza per impostazione predefinita una versione inferiore se necessario.

Verifica la versione di Open e SSL TLS

Per scaricare la versione di Open SSL utilizzata da Node.js sul tuo computer, esegui il comando seguente.

```
node -p process.versions
```

La versione di Open SSL nell'elenco è la versione utilizzata da Node.js, come illustrato nell'esempio seguente.

```
openssl: '1.1.1b'
```

Per scaricare la versione di Node.js TLS usata da sul tuo computer, avvia la shell Node ed esegui i seguenti comandi, nell'ordine.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
```

```
> tlsSocket.getProtocol();
```

L'ultimo comando restituisce la TLS versione, come illustrato nell'esempio seguente.

```
'TLSv1.3'
```

Per impostazione predefinitaTLS, Node.js utilizza questa versione di e tenta di negoziare un'altra versione TLS se una chiamata non ha esito positivo.

Applica una versione minima di TLS

Node.js negozia una versione di TLS quando una chiamata fallisce. È possibile applicare la TLS versione minima consentita durante questa negoziazione, sia quando si esegue uno script dalla riga di comando sia per richiesta nel codice. JavaScript

Per specificare la TLS versione 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 far sì che la versione TLS 1.2 sia la versione minima consentita, specificate l'`--tls-min-v1.2`argomento durante l'esecuzione dello script, come illustrato nell'esempio seguente.

```
node --tls-min-v1.2 yourScript.js
```

Per specificare la TLS versione 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 la versione 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 TLS versione minima consentita per una richiesta specifica nel JavaScript codice, utilizzate il `http0ptions` 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'
            }
        )
    })
});
```

Verifica e applica TLS in uno script del browser

Quando si utilizza il form SDK JavaScript in uno script del browser, le impostazioni del browser controllano la versione dello TLS script utilizzata. La versione TLS utilizzata dal browser non può

essere rilevata o impostata tramite script e deve essere configurata dall'utente. Per verificare e applicare la versione TLS utilizzata in uno script del browser, fate riferimento alle istruzioni del 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 Windows, digita *`Internet options`*.
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 Utente 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 1.2 come predefinito. TLS
5. Fai clic su OK.
6. Chiudi il browser e riavvia Mozilla Firefox.

Apple Safari

Non ci sono opzioni per abilitare i SSL protocolli. Se utilizzi la versione 7 o successiva di Safari, la TLS 1.2 viene abilitata automaticamente.

Migrare dalla versione 2.x alla 3.x del AWS SDK for JavaScript

La AWS SDK for JavaScript versione 3 è una riscrittura importante della versione 2. La sezione descrive le differenze tra le due versioni e spiega come migrare dalla versione 2 alla versione 3 del SDK for. JavaScript

Migra il tuo codice a SDK for JavaScript v3 usando codemod

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 la v3. APIs 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 });
await client.listTables({}).promise()
  .then(console.log)
  .catch(console.error);
```

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 });
await client.listTables()
  .then(console.log)
  .catch(console.error);
```

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.

Cosa c'è di nuovo nella versione 3

La versione 3 di 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, SDK è scritto, il che presenta molti vantaggi TypeScript, come la digitazione statica.

> **Important**
>
> Gli esempi di codice per la v3 in questa guida sono scritti in ECMAScript 6 ()ES6. ES6offre una nuova sintassi e nuove funzionalità per rendere il codice più moderno e leggibile e

fare di più. ES6richiede l'utilizzo della versione 13.x o successiva di Node.js. 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 di 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 AWS servizi. Tuttavia, se è necessario utilizzare solo alcuni AWS servizi, è necessario aumentare le dimensioni dell'applicazione con codice non necessario o non utilizzato.

Nella 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);
  }
});
```

v3 ha il seguente aspetto.

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

const dbclient = new DynamoDBClient({ region: "us-west-2" });

try {
  const results = await dbclient.send(new ListTablesCommand);

  for (const item of results.TableNames) {
    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 nella 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 utilizzando 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" });
const tableParams = {
  TableName: TABLE_NAME
};

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 nel SDK modulo 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 modello 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 dynamoDB = new DynamoDB({ region: 'us-west-2' });
var tableParams = {
  TableName: TABLE_NAME
};
async function run() => {
  try {
    const data = await dynamoDB.createTable(tableParams);
    console.log("Success", data);
  }
```

```
  catch (err) {
    console.log("Error", err);
  }
};
run();
```

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

Nuovo stack di middleware

la v2 of the ti SDK ha consentito 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.

Nella 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

chiamarecontext, e che è un oggetto che contiene alcune informazioni sull'operazione chiamata. La funzione restituisce una funzione che accettaargs, 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);
  },
  {
    name: "my-middleware",
    override: true,
    step: "build"
  }
);

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

Cosa c'è di diverso tra la AWS SDK for JavaScript v2 e la v3

Questa sezione riporta le modifiche importanti dalla AWS SDK for JavaScript v2 alla v3. Poiché v3 è una riscrittura modulare della v2, alcuni concetti di base sono diversi tra v2 e v3. Puoi scoprire queste modifiche nei post del nostro blog. I seguenti post del blog ti aiuteranno a tenerti aggiornato:

- Pacchetti modulari in AWS SDK for JavaScript
- Presentazione di Middleware Stack in Modular AWS SDK for JavaScript

Di seguito è riportato il riepilogo delle modifiche all'interfaccia dalla AWS SDK for JavaScript v2 alla v3. L'obiettivo è aiutarti a trovare facilmente gli equivalenti v3 delle API v2 che già conosci.

Argomenti

- Costruttori clienti
- Fornitori di credenziali
- Considerazioni su Amazon S3
- Client per documenti DynamoDB
- Camerieri e firmatari

- Note su clienti di servizi specifici

Costruttori clienti

Questo elenco è indicizzato in base ai parametri di configurazione v2.

- `computeChecksums`
  - v2: se calcolare i MD5 checksum per i corpi di payload quando il servizio li accetta (attualmente supportato solo in S3).
  - v3: i comandi applicabili di S3 (PutObject PutBucketCors, ecc.) calcoleranno automaticamente i checksum per il payload della richiesta. MD5 È inoltre possibile specificare un algoritmo di checksum diverso nel parametro dei comandi `ChecksumAlgorithm` per utilizzare un algoritmo di checksum diverso. Puoi trovare ulteriori informazioni nell'annuncio delle funzionalità di S3.
- `convertResponseTypes`
  - v2: Se i tipi vengono convertiti durante l'analisi dei dati di risposta.
  - v3: obsoleto. Questa opzione è considerata non sicura dai tipi perché non converte tipi come time stamp o binari base64 dalla risposta. JSON
- `correctClockSkew`
  - v2: Se applicare una correzione dell'inclinazione dell'orologio e riprovare le richieste che falliscono a causa di un orologio del client distorto.
  - v3: obsoleto. SDKapplica sempre una correzione dell'inclinazione dell'orologio.
- `systemClockOffset`
  - v2: Un valore di offset in millisecondi da applicare a tutti gli orari di firma.
  - v3: nessuna modifica.
- `credentials`
  - v2: Le AWS credenziali con cui firmare le richieste.
  - v3: Nessuna modifica. Può anche essere una funzione asincrona che restituisce le credenziali. Se la funzione restituisce un`expiration (Date)`, la funzione verrà richiamata nuovamente all'avvicinarsi della data e dell'ora di scadenza. Vedi il APIriferimento v3 per le credenziali. `AwsAuthInputConfig`
- `endpointCacheSize`
  - v2: La dimensione della cache globale che archivia gli endpoint derivanti dalle operazioni di rilevamento degli endpoint.
  - v3: Nessuna modifica.
- `endpointDiscoveryEnabled`

  - v2: se chiamare dinamicamente le operazioni con gli endpoint forniti dal servizio.
  - v3: Nessuna modifica.
- `hostPrefixEnabled`
  - v2: Se marshallare i parametri della richiesta al prefisso del nome host.
  - v3: obsoleto. SDKinserisce sempre il prefisso del nome host quando necessario.
- `httpOptions`

  Un insieme di opzioni da passare alla richiesta di basso livello. HTTP Queste opzioni sono aggregate in modo diverso nella v3. Puoi configurarle fornendone una nuova. `requestHandler` Ecco l'esempio di impostazione delle opzioni http nel runtime di Node.js. Puoi trovare ulteriori informazioni nella versione di APIriferimento v3 per NodeHttpHandler.

  Tutte le richieste v3 vengono utilizzate per impostazione HTTPS predefinita. Devi solo fornire dati personalizzatihttpsAgent.

```
const { Agent } = require("https");
const { Agent: HttpAgent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpsAgent: new Agent({
            /*params*/
        }),
        connectionTimeout: /*number in milliseconds*/,
        socketTimeout: /*number in milliseconds*/
    }),
});
```

  Se stai passando un endpoint personalizzato che utilizza http, devi fornirehttpAgent.

```
const { Agent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");

const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: new Agent({
            /*params*/
        }),
    }),
    endpoint: "http://example.com",
```

```
});
```

Se il client è in esecuzione nei browser, è disponibile un diverso set di opzioni. Puoi trovare ulteriori informazioni nella versione di APIriferimento v3 per FetchHttpHandler.

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
const dynamodbClient = new DynamoDBClient({
    requestHandler: new FetchHttpHandler({
        requestTimeout: /* number in milliseconds */
    }),
});
```

Ogni opzione di `httpOptions` è specificata di seguito:

- `proxy`
  - v2: Da inviare tramite proxy URL alle richieste.
  - v3: È possibile configurare un proxy con un agente seguendo la procedura Configurazione dei proxy per Node.js.
- `agent`
  - v2: L'oggetto Agent con cui eseguire le richieste. HTTP Utilizzato per il pool di connessioni.
  - v3: È possibile configurare `httpAgent` o `httpsAgent` come mostrato negli esempi precedenti.
- `connectTimeout`
  - v2: Imposta il timeout del socket dopo che non è riuscito a stabilire una connessione con il server dopo millisecondi. `connectTimeout`
  - v3: è disponibile nelle opzioni. `connectionTimeoutNodeHttpHandler`
- `timeout`
  - v2: Il numero di millisecondi che una richiesta può impiegare prima di essere terminata automaticamente.
  - v3: è disponibile nelle opzioni. `socketTimeoutNodeHttpHandler`
- `xhrAsync`
  - v2: se SDK invierà richieste HTTP asincrone.
  - v3: obsoleto. Le richieste sono sempre asincrone.
- `xhrWithCredentials`
  - v2: Imposta la proprietà "withCredentials" di un oggetto. XMLHttpRequest
  - v3: Non disponibile. SDKeredita le configurazioni di recupero predefinite.
- `logger`

  - v2: Un oggetto che risponde `.write()` (come uno stream) o `.log()` (come l'oggetto console) per registrare le informazioni sulle richieste.
  - v3: Nessuna modifica. Nella v3 sono disponibili log più granulari.
- `maxRedirects`
  - v2: Il numero massimo di reindirizzamenti da seguire per una richiesta di servizio.
  - v3: obsoleto. SDKnon segue i reindirizzamenti per evitare richieste involontarie tra regioni.
- `maxRetries`
  - v2: Il numero massimo di tentativi da eseguire per una richiesta di servizio.
  - v3: Modificato in. `maxAttempts` Vedi di più nel APIriferimento alla v3 per. RetryInputConfig Nota che `maxAttempts` dovrebbe esser`maxRetries + 1`.
- `paramValidation`
  - v2: Se i parametri di input devono essere convalidati rispetto alla descrizione dell'operazione prima di inviare la richiesta.
  - v3: obsoleto. SDKnon esegue la convalida sul lato client in fase di esecuzione.
- `region`
  - v2: La regione a cui inviare le richieste di servizio.
  - v3: Nessuna modifica. Può anche essere una funzione asincrona che restituisce una stringa di regione.
- `retryDelayOptions`
  - v2: Una serie di opzioni per configurare il ritardo tra i tentativi in caso di errori riproducibili.
  - v3: obsoleto. SDKsupporta una strategia di riprova più flessibile con l'opzione Client Constructor. `retryStrategy` Scopri di più nel riferimento alla versione 3 API.
- `s3BucketEndpoint`
  - v2: Se l'endpoint fornito si rivolge a un singolo bucket (falso se si rivolge all'endpoint root). API
  - v3: Modificato in. `bucketEndpoint` Vedi di più nel APIriferimento alla v3 per. bucketEndpoint Nota che se impostato su`true`, si specifica l'endpoint della richiesta nel parametro di `Bucket` richiesta, l'endpoint originale verrà sovrascritto. Mentre nella v2, l'endpoint di richiesta in client constructor sovrascrive il parametro di richiesta. `Bucket`
- `s3DisableBodySigning`
  - v2: se disabilitare la firma del corpo di S3 quando si utilizza la versione di firma v4.
  - v3: rinominato in. `applyChecksum`
- `s3ForcePathStyle`
  - v2: Se forzare lo stile del percorso URLs per gli oggetti S3.
  - v3: rinominato in. `forcePathStyle`
- `s3UseArnRegion`

  - v2: se sovrascrivere la regione di richiesta con la regione dedotta dalle risorse richieste. ARN
  - v3: rinominato in. `useArnRegion`
- `s3UsEast1RegionalEndpoint`
  - v2: Quando la regione è impostata su «us-east-1», se inviare la richiesta s3 agli endpoint globali o agli endpoint regionali «us-east-1».
  - v3: obsoleto. Il client S3 utilizzerà sempre l'endpoint regionale se la regione è impostata su. `us-east-1` Puoi impostare la regione per inviare richieste `aws-global` all'endpoint globale S3.
- `signatureCache`
  - v2: Se la firma con cui firmare le richieste (sovrascrivendo la API configurazione) è memorizzata nella cache.
  - v3: obsoleto. SDKmemorizza sempre nella cache le chiavi di firma con hash.
- `signatureVersion`
  - v2: La versione della firma con cui firmare le richieste (sovrascrivendo la configurazione). API
  - v3: obsoleto. La firma V2 supportata nella v2 SDK è stata deprecata da. v3 supporta solo la firma v4. AWS
- `sslEnabled`
  - v2: se è abilitato per le richieste. SSL
  - v3: Rinominato in. `tls`
- `stsRegionalEndpoints`
  - v2: se inviare la richiesta sts agli endpoint globali o agli endpoint regionali.
  - v3: obsoleto. STSil client utilizzerà sempre gli endpoint regionali se impostato su una regione specifica. È possibile impostare la regione per `aws-global` inviare la richiesta all'endpoint STS globale.
- `useAccelerateEndpoint`
  - v2: Se utilizzare l'endpoint Accelerate con il servizio S3.
  - v3: Nessuna modifica.

Fornitori di credenziali

Nella v2, l'SDK for JavaScript fornisce un elenco di provider di credenziali tra cui scegliere, oltre a una catena di fornitori di credenziali, disponibile per impostazione predefinita su Node.js, che tenta di caricare AWS le credenziali da tutti i provider più comuni. L'SDK per la versione JavaScript 3 semplifica l'interfaccia del provider di credenziali, semplificando l'utilizzo e la scrittura di provider di credenziali personalizzati. Oltre a una nuova catena di fornitori di credenziali, l'SDK per la versione

JavaScript 3 fornisce tutti un elenco di provider di credenziali che mirano a fornire una soluzione equivalente alla versione 2.

Ecco tutti i provider di credenziali nella v2 e i loro equivalenti nella v3.

Provider di credenziali predefinito

Il provider di credenziali predefinito è il modo in cui l'SDK JavaScript risolve le AWS credenziali se non ne fornite una esplicitamente.

- v2: CredentialProviderChainin Node.js risolve le credenziali dai sorgenti nel seguente ordine:
  - Variabile ambientale
  - File di credenziali condiviso
  - Credenziali del contenitore ECS
  - Processo esterno di generazione
  - Token OIDC dal file specificato
  - Metadati dell'istanza EC2

  Se uno dei provider di credenziali sopra indicati non riesce a risolvere la AWS credenziale, la catena passa al provider successivo finché non viene risolta una credenziale valida e la catena genererà un errore quando tutti i provider falliscono.

  Nei runtime Browser e React Native, la catena di credenziali è vuota e le credenziali devono essere impostate in modo esplicito.
- v3: DefaultProvider. Le fonti delle credenziali e l'ordine di fallback non cambiano nella v3. Supporta anche le credenziali.AWS IAM Identity Center

Credenziali temporanee

- v2: `ChainableTemporaryCredentials`rappresenta le credenziali temporanee recuperate da. `AWS.STS` Senza parametri aggiuntivi, le credenziali verranno recuperate dall'operazione. `AWS.STS.getSessionToken()` Se viene fornito un ruolo IAM, l'`AWS.STS.assumeRole()`operazione verrà invece utilizzata per recuperare le credenziali per il ruolo. `AWS.ChainableTemporaryCredentials`differisce dal modo `AWS.TemporaryCredentials` in cui vengono gestiti MasterCredentials e gli aggiornamenti. `AWS.ChainableTemporaryCredentials`aggiorna le credenziali scadute utilizzando le MasterCredentials passate dall'utente per supportare il concatenamento delle credenziali STS. Tuttavia, comprime `AWS.TemporaryCredentials` ricorsivamente le MasterCredentials durante

l'istanziazione, precludendo la possibilità di aggiornare le credenziali che richiedono credenziali intermedie e temporanee.

L'**TemporaryCredentials**originale è stato `ChainableTemporaryCredentials` deprecato a favore di in v2.

- v3:. **fromTemporaryCredentials** Puoi chiamare `fromTemporaryCredentials()` dal `@aws-sdk/credential-providers` pacchetto. Ecco un esempio:

```
import { FooClient } from "@aws-sdk/client-foo";
import { fromTemporaryCredentials } from "@aws-sdk/credential-providers"; // ES6
 import
// const { FooClient } = require("@aws-sdk/client-foo");
// const { fromTemporaryCredentials } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const sourceCredentials = {
  // A credential can be a credential object or an async function that returns a
 credential object
};
const client = new FooClient({
  credentials: fromTemporaryCredentials({
    masterCredentials: sourceCredentials,
    params: { RoleArn },
  }),
});
```

Credenziali di identità Amazon Cognito

Carica le credenziali dal servizio Amazon Cognito Identity, normalmente utilizzato nei browser.

- v2: `CognitoIdentityCredentials`rappresenta le credenziali recuperate da STS Web Identity Federation utilizzando il servizio Amazon Cognito Identity.
- v3: `Cognito Identity Credential Provider`Il `@aws/credential-providers`pacchetto fornisce due funzioni di fornitore di credenziali, una delle quali `fromCognitoIdentity`richiede un ID di identità e chiama`cognitoIdentity:GetCredentialsForIdentity`, mentre l'altra `fromCognitoIdentityPool`accetta un ID di pool di identità, chiama alla prima chiamata e `cognitoIdentity:GetId` poi chiama. `fromCognitoIdentity` Le successive invocazioni di quest'ultimo non vengono reinvocate. GetId

Il provider implementa il «Simplified Flow» descritto nella Amazon Cognito Developer Guide. Il «Classic Flow» che prevede la chiamata *`cognito:GetOpenIdToken`* e quindi la chiamata *`sts:AssumeRoleWithWebIdentity`* non è supportato. Se ne hai bisogno, apri una richiesta di funzionalità.

```
// fromCognitoIdentityPool example
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers"; // ES6
 import
// const { fromCognitoIdentityPool } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

```
// fromCognitoIdentity example
import { fromCognitoIdentity } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromCognitoIdentity } = require("@aws-sdk/credential-provider-cognito-
identity"); // CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentity({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityId: "us-east-1:128d0a74-c82f-4553-916d-90053e4a8b0f",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
```

```
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

Credenziale EC2 Metadata (IMDS)

Rappresenta le credenziali ricevute dal servizio di metadati su un'istanza Amazon EC2.

- v2: **`EC2MetadataCredentials`**
- v3: `fromInstanceMetadata`: Crea un provider di credenziali che raccoglierà le credenziali dall'Amazon EC2 Instance Metadata Service.

```
import { fromInstanceMetadata } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromInstanceMetadata } = require("@aws-sdk/credential-providers"); //
 CommonJS import

const client = new FooClient({
  credentials: fromInstanceMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

Credenziali ECS

Rappresenta le credenziali ricevute dall'URL specificato. Questo provider richiederà credenziali temporanee all'URI specificato dalla variabile di `AWS_CONTAINER_CREDENTIALS_FULL_URI` ambiente `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` o.

- v2: `ECSCredentials` o. `RemoteCredentials`
- v3: `fromContainerMetadata`crea un provider di credenziali che fornirà le credenziali da Amazon ECS Container Metadata Service.

```
import { fromContainerMetadata } from "@aws-sdk/credential-providers"; // ES6 import

const client = new FooClient({
  credentials: fromContainerMetadata({
    maxRetries: 3, // Optional
```

```
    timeout: 0, // Optional
  }),
});
```

Credenziali del file system

- v2: `FileSystemCredentials`rappresenta le credenziali di un file JSON su disco.
- v3: obsoleto. È possibile leggere esplicitamente il file JSON e fornirlo al client. Se ne hai bisogno, apri una richiesta di funzionalità.

Provider di credenziali SAML

- v2: `SAMLCredentials`rappresenta le credenziali recuperate dal supporto SAML di STS.
- v3: non disponibile. Se ne hai bisogno, apri una richiesta di funzionalità.

Credenziali condivise: credenziali (file)

Carica le credenziali dal file di credenziali condiviso (predefinito `~/.aws/credentials` o definito dalla variabile di ambiente). `AWS_SHARED_CREDENTIALS_FILE` Questo file è supportato da diversi SDK e strumenti. AWS Puoi fare riferimento al documento sui file di configurazione e credenziali condivisi per ulteriori informazioni.

- v2: **`SharedIniFileCredentials`**
- v3:. `fromIni`

```
import { fromIni } from "@aws-sdk/credential-providers";
// const { fromIni } from("@aws-sdk/credential-providers");

const client = new FooClient({
  credentials: fromIni({
    configFilepath: "~/.aws/config", // Optional
    filepath: "~/.aws/credentials", // Optional
    mfaCodeProvider: async (mfaSerial) => {
      // implement a pop-up asking for MFA code
      return "some_code";
    }, // Optional
    profile: "default", // Optional
    clientConfig: { region }, // Optional
```

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

Credenziali di identità Web

Recupera le credenziali utilizzando il token OIDC da un file su disco. È comunemente usato in EKS.

- v2: `TokenFileWebIdentityCredentials`.
- v3: **`fromTokenFile`**

```
import { fromTokenFile } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromTokenFile } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromTokenFile({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Credenziali Web Identity Federation

Recupera le credenziali dal supporto STS Web Identity Federation.

- v2: **`WebIdentityCredentials`**
- v3: **`fromWebToken`**

```
import { fromWebToken } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromWebToken } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromWebToken({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
```

```
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Considerazioni su Amazon S3

Caricamento multiparte di Amazon S3

Nella versione 2, il client Amazon S3 contiene `upload()`un'operazione che supporta il caricamento di oggetti di grandi dimensioni con la funzionalità di caricamento multiparte offerta da Amazon S3.

Nella v3, il pacchetto è disponibile. `@aws-sdk/lib-storage` Supporta tutte le funzionalità offerte nell'`upload()`operazione v2 e supporta sia Node.js che il runtime del browser.

URL predefinito Amazon S3

Nella versione 2, il client Amazon S3 contiene `getSignedUrl()`le operazioni `getSignedUrlPromise()`e per generare un URL che gli utenti possono utilizzare per caricare o scaricare oggetti da Amazon S3.

Nella v3, il pacchetto è disponibile. `@aws-sdk/s3-request-presigner` Questo pacchetto contiene le funzioni per entrambe `getSignedUrl()` le `getSignedUrlPromise()` operazioni. Questo post del blog illustra i dettagli di questo pacchetto.

Reindirizzamenti regionali Amazon S3

Se viene passata una regione errata al client Amazon S3 e viene generato un errore successivo `PermanentRedirect` (stato 301), il client Amazon S3 nella v3 supporta i reindirizzamenti regionali (precedentemente noti come Amazon S3 Global Client nella v2). Puoi utilizzare il `followRegionRedirects`flag nella configurazione del client per fare in modo che il client Amazon S3 segua i reindirizzamenti regionali e supporti la sua funzione di client globale.

> **Note**
>
> Tieni presente che questa funzionalità può comportare una latenza aggiuntiva poiché le richieste non riuscite vengono ritentate con una regione corretta quando viene ricevuto un

errore con lo stato 301. `PermanentRedirect` Questa funzione deve essere utilizzata solo se non conosci in anticipo la regione dei tuoi bucket.

Streaming Amazon S3 e risposte bufferizzate

L'SDK v3 preferisce non bufferizzare risposte potenzialmente di grandi dimensioni. Questo si verifica comunemente nell'`GetObject`operazione Amazon S3, che ha restituito `Buffer` a nella v2, ma restituisce una `Stream` nella v3.

Per Node.js, è necessario utilizzare lo stream o il garbage collect del client o del relativo gestore di richieste per mantenere le connessioni aperte a nuovo traffico liberando i socket.

```
// v2
const get = await s3.getObject({ ... }).promise(); // this buffers consumes the stream
 already.
```

```
// v3, consume the stream to free the socket
const get = await s3.getObject({ ... }); // object .Body has unconsumed stream
const str = await get.Body.transformToString(); // consumes the stream

// other ways to consume the stream include writing it to a file,
// passing it to another consumer like an upload, or buffering to
// a string or byte array.
```

Per ulteriori informazioni, consulta la sezione sull'esaurimento dei socket.

Client per documenti DynamoDB

Utilizzo di base del client di documenti DynamoDB nella v3

- Nella v2, puoi usare la `AWS.DynamoDB.DocumentClient`classe per chiamare le API DynamoDB con tipi JavaScript nativi come Array, Number e Object. Semplifica quindi l'utilizzo degli elementi in Amazon DynamoDB astraendo la nozione di valori di attributo.
- Nella v3, è disponibile il client equivalente. `@aws-sdk/lib-dynamodb` È simile ai normali client di servizio di v3 SDK, con la differenza che nel costruttore richiede un client DynamoDB di base.

Esempio:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb"; // ES6 import
// const { DynamoDBClient } = require("@aws-sdk/client-dynamodb"); // CommonJS import
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb"; // ES6
 import
// const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb"); //
 CommonJS import

// Bare-bones DynamoDB Client
const client = new DynamoDBClient({});

// Bare-bones document client
const ddbDocClient = DynamoDBDocumentClient.from(client); // client is DynamoDB client

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "1",
      content: "content from DynamoDBDocumentClient",
    },
  })
);
```

`Undefined`valori inseriti durante il marshalling

- Nella v2, `undefined` i valori negli oggetti venivano automaticamente omessi durante il processo di marshalling in DynamoDB.
- Nella v3, il comportamento di marshalling predefinito in è cambiato: gli oggetti con valori non vengono più omessi`@aws-sdk/lib-dynamodb`. `undefined` Per allinearsi alle funzionalità della v2, gli sviluppatori devono impostare esplicitamente `removeUndefinedValues` to `true` nel DynamoDB `marshallOptions` Document Client.

Esempio:

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

const client = new DynamoDBClient({});

// The DynamoDBDocumentClient is configured to handle undefined values properly
const ddbDocClient = DynamoDBDocumentClient.from(client, {
```

```
  marshallOptions: {
    removeUndefinedValues: true
  }
});

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "123",
      content: undefined // This value will be automatically omitted
    };
  })
);
```

Altri esempi e configurazioni sono disponibili nel pacchetto README.

Camerieri e firmatari

Questa pagina descrive l'utilizzo di camerieri e firmatari nella v3. AWS SDK for JavaScript

Waiter

Nella v2, tutti i camerieri sono legati alla classe service client ed è necessario specificare nell'input del cameriere quale stato di progettazione il cliente aspetterà. Ad esempio, è necessario chiamare `waitFor("bucketExists")`per attendere che un bucket appena creato sia pronto.

Nella v3, non è necessario importare i camerieri se l'applicazione non ne ha bisogno. Inoltre, puoi importare solo il cameriere necessario per attendere lo stato particolare desiderato. In questo modo, puoi ridurre le dimensioni del pacchetto e migliorare le prestazioni. Ecco un esempio di attesa che il bucket sia pronto dopo la creazione:

```
import { S3Client, CreateBucketCommand, waitUntilBucketExists } from "@aws-sdk/client-
s3"; // ES6 import
// const { S3Client, CreateBucketCommand, waitUntilBucketExists } = require("@aws-sdk/
client-s3"); // CommonJS import

const Bucket = "BUCKET_NAME";
const client = new S3Client({ region: "REGION" });
const command = new CreateBucketCommand({ Bucket });

await client.send(command);
```

```
await waitUntilBucketExists({ client, maxWaitTime: 60 }, { Bucket });
```

Puoi trovare tutto su come configurare i camerieri nel post del blog dedicato ai camerieri nella v3. AWS SDK for JavaScript

Amazon CloudFront Firmatario

Nella v2, puoi firmare la richiesta di accesso alle CloudFront distribuzioni Amazon con restrizioni. `AWS.CloudFront.Signer`

Nella v3, hai le stesse utilità fornite nel pacchetto. `@aws-sdk/cloudfront-signer`

Amazon RDS Signer

Nella v2, puoi generare il token di autenticazione su un database Amazon RDS utilizzando. `AWS.RDS.Signer`

Nella v3, la classe di utilità simile è disponibile in pacchetto. `@aws-sdk/rds-signer`

Firmatario Amazon Polly

Nella v2, puoi generare un URL firmato per il discorso sintetizzato dal servizio Amazon Polly con. `AWS.Polly.Presigner`

Nella v3, la funzione di utilità simile è disponibile nel pacchetto. `@aws-sdk/polly-request-presigner`

Note su clienti di servizi specifici

AWS Lambda

Il tipo di risposta alle chiamate Lambda è diverso in v2 e v3.

```
// v2
import { Lambda } from "@aws-sdk/client-lambda";
import AWS from "aws-sdk";

const lambda = new AWS.Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
```

```
  Payload: JSON.stringify({ message: "hello" }),
}).promise();

// in v2, Lambda::invoke::Payload is automatically converted to string via a
// specific code customization.
const payloadIsString = typeof invoke.Payload === "string";
console.log("Invoke response payload type is string:", payloadIsString);

const payloadObject = JSON.parse(invoke.Payload);
console.log("Invoke response object", payloadObject);
```

```
// v3
const lambda = new Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
});

// in v3, Lambda::invoke::Payload is not automatically converted to a string.
// This is to reduce the number of customizations that create inconsistent behaviors.
const payloadIsByteArray = invoke.Payload instanceof Uint8Array;
console.log("Invoke response payload type is Uint8Array:", payloadIsByteArray);

// To maintain the old functionality, only one additional method call is needed:
// v3 adds a method to the Uint8Array called transformToString.
const payloadObject = JSON.parse(invoke.Payload.transformToString());
console.log("Invoke response object", payloadObject);
```

Amazon SQS

Checksum MD5

Per ignorare il calcolo dei checksum MD5 dei corpi dei messaggi, *md5* impostate su false nell'oggetto di configurazione. Altrimenti, per impostazione predefinita, l'SDK calcolerà il checksum per l'invio dei messaggi e convaliderà il checksum per i messaggi recuperati.

```
// Example: Skip MD5 checksum in Amazon SQS
import { SQS } from "@aws-sdk/client-sqs";

new SQS({
  md5: false // note: only available in v3.547.0 and higher
});
```

Quando si utilizzava una `QueueUrl` soluzione personalizzata nelle operazioni Amazon SQS che lo utilizzavano come parametro di input, nella versione 2 era possibile fornire una soluzione personalizzata `QueueUrl` che sostituisse l'endpoint predefinito del client Amazon SQS.

Messaggi multiregionali

È necessario utilizzare un client per regione nella v3. La AWS regione è pensata per essere inizializzata a livello di client e non modificata tra le richieste.

```
import { SQS } from "@aws-sdk/client-sqs";

const sqsClients = {
  "us-east-1": new SQS({ region: "us-east-1" }),
  "us-west-2": new SQS({ region: "us-west-2" }),
};

const queues = [
  { region: "us-east-1", url: "https://sqs.us-east-1.amazonaws.com/{AWS_ACCOUNT}/
MyQueue" },
  { region: "us-west-2", url: "https://sqs.us-west-2.amazonaws.com/{AWS_ACCOUNT}/
MyOtherQueue" },
];

for (const { region, url } of queues) {
  const params = {
    MessageBody: "Hello",
    QueueUrl: url,
  };
  await sqsClients[region].sendMessage(params);
}
```

Endpoint personalizzato

Nella versione 3, quando si utilizza un endpoint personalizzato, ovvero uno diverso dagli endpoint Amazon SQS pubblici predefiniti, è necessario impostare sempre l'endpoint sul client Amazon SQS oltre che sul campo. `QueueUrl`

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  // client endpoint should be specified in v3 when not the default public SQS endpoint
 for your region.
```

```
  // This is required for versions <= v3.506.0
  // This is optional but recommended for versions >= v3.507.0 (a warning will be
 emitted)
  endpoint: "https://my-custom-endpoint:8000/",
});

await sqs.sendMessage({
  QueueUrl: "https://my-custom-endpoint:8000/1234567/MyQueue",
  Message: "hello",
});
```

Se non utilizzi un endpoint personalizzato, non è necessario impostarlo sul client. `endpoint`

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  region: "us-west-2",
});

await sqs.sendMessage({
  QueueUrl: "https://sqs.us-west-2.amazonaws.com/1234567/MyQueue",
  Message: "hello",
});
```

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 iscriverti a un RSS feed.

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 AWS SDKs e guida di riferimento agli strumenti.	22 agosto 2022
Applica una versione minima TLS	Sono state aggiunte informazioni su TLS 1.3.	31 marzo 2022
AWS Lambda Tutorial aggiornato	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 Node.js per AWS SDK for JavaScript V3.	20 ottobre 2020

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