Guida per gli sviluppatori per la versione SDK 3

AWS SDK for JavaScript

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

Table of Contents

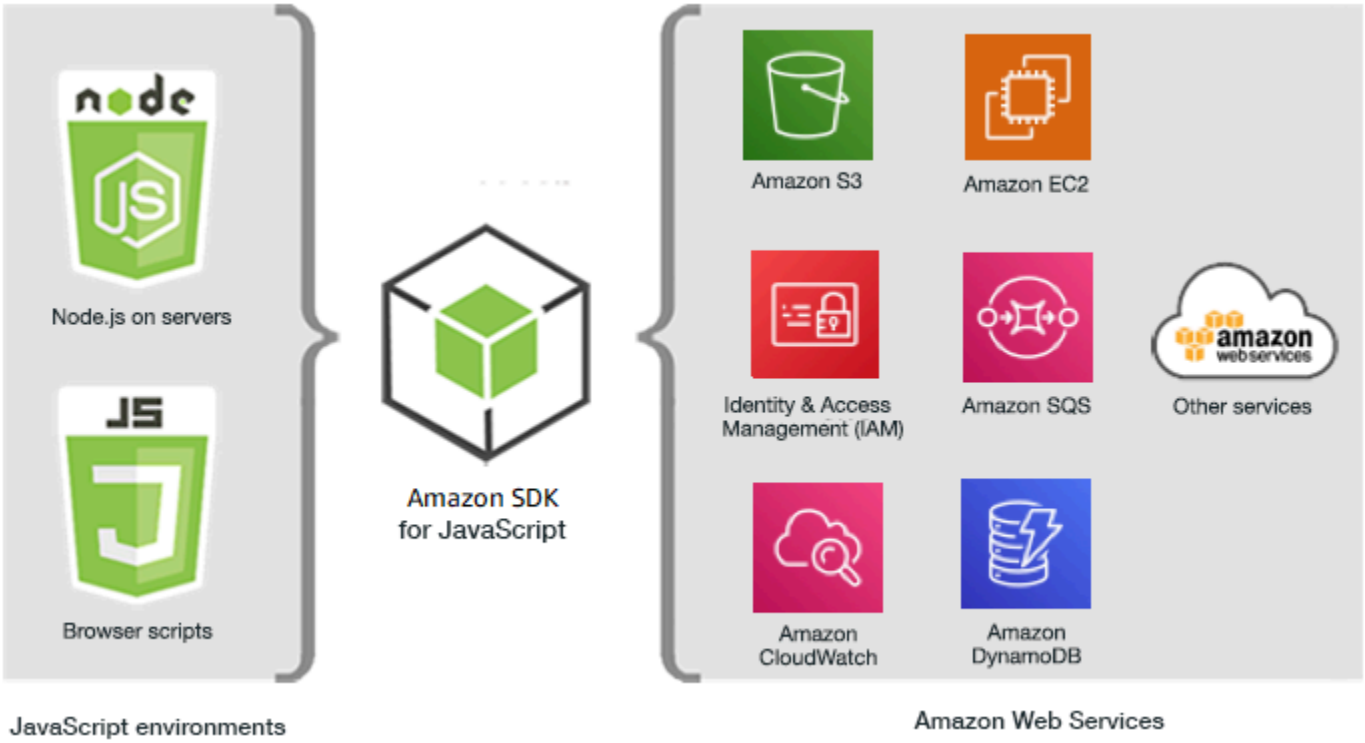
La AWS SDK for JavaScript V3 API Reference Guide descrive in dettaglio tutte le operazioni API per la AWS SDK for JavaScript versione 3 (V3).

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

Qual è 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. Nozioni di base

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. Servizi AWSSDK per esempi di JavaScript codice (v3)

Manutenzione e supporto per le versioni principali dell'SDK

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

- AWS SDKs e politica di manutenzione degli strumenti
- AWS SDKs e matrice di supporto delle versioni 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 APIs differisce tra Node.js e il browser, evidenziamo tali differenze.

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 particolari APIs 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.

- AWS Lambda Usalo per incapsulare la 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 SDKs e Guida di riferimento agli strumenti: contiene impostazioni, funzionalità e altri concetti fondamentali comuni a. AWS SDKs
- JavaScript Blog per sviluppatori
- AWS Re: post
- JavaScript esempi nella Code Library AWS
- AWS Repository di esempi di codice
- Canale Gitter
- Stack Overflow
- Domande Stack Overflow etichettate AWS -sdk-js
- GitHub
  - Fonte SDK
  - Fonte della documentazione

Inizia a usare l' AWS SDK per JavaScript

L' 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' AWS SDK for JavaScript in ciascuno di questi JavaScript ambienti.

Argomenti

Autenticazione SDK con AWS

È necessario stabilire in che modo il codice si autentica AWS durante lo sviluppo con. Servizi AWSÈ 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 strumenti AWS SDKs 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 AWS IAM Identity Center configurarlo. 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 AWS SDKs and Tools Reference Guide:

- 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 Posizione dei file condivisi nella Guida di riferimento agli strumenti AWS SDKs e strumenti.
- Il `config` file condiviso imposta l'`region`impostazione. Questo imposta l'impostazione predefinita Regione AWS utilizzata dall'SDK per AWS le richieste. Questa regione viene utilizzata per le richieste di servizio SDK che non sono specificate con una regione da utilizzare.

- L'SDK utilizza la configurazione del provider di token SSO del profilo per acquisire le credenziali prima di inviare richieste a. AWS Il `sso_role_name` valore, che è un ruolo IAM connesso a un set di autorizzazioni IAM Identity Center, consente l'accesso ai dati Servizi AWS utilizzati nell'applicazione.

  Il seguente `config` file di esempio mostra un profilo predefinito impostato con la configurazione del provider di token SSO. L'`sso_session`impostazione del profilo si riferisce alla `sso-session`sezione denominata. La `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
```

L' AWS SDK per JavaScript v3 non richiede l'aggiunta di pacchetti aggiuntivi (come `SSO` e`SSOOIDC`) 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 Servizi AWS, è necessaria una sessione attiva del portale di AWS accesso affinché l'SDK utilizzi l'autenticazione IAM Identity Center per risolvere le credenziali. A seconda della durata della sessione configurata, l'accesso alla fine scadrà e l'SDK riscontrerà un errore di autenticazione. Per accedere al portale di 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.

Utilizzo delle credenziali di accesso alla console

È possibile utilizzare le credenziali di accesso esistenti AWS della Console di gestione per l'accesso programmatico ai servizi. AWS Dopo un flusso di autenticazione basato su browser, AWS genera credenziali temporanee che funzionano con strumenti di sviluppo locali come AWS CLI e SDK for. AWS JavaScript Questa funzionalità semplifica il processo di configurazione e gestione delle credenziali CLI AWS . Per informazioni su come iniziare, segui le istruzioni per accedere allo sviluppo AWS locale utilizzando le credenziali della console.

Quando esegui il aws login comando, puoi selezionare una delle sessioni attive della console o accedere tramite il flusso di autenticazione basato sul browser: questo genererà automaticamente credenziali temporanee. L' AWS SDK aggiorna JavaScript automaticamente le credenziali 5 minuti prima della scadenza, con ogni set di credenziali valido per un massimo di 12 ore. Per ulteriori informazioni, consulta `fromLoginCredentials()`.

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 per JavaScript V3, consulta Fornitori di credenziali standardizzati nella and Tools Reference Guide.AWS SDKs

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. AWS consiglia di utilizzare la versione Active LTS di Node.js per lo sviluppo.

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 "node: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 "node: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' AWS 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' AWS SDK per, consulta. JavaScript SDK per esempi di JavaScript codice (v3)

Questo esempio di applicazione web mostra:

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

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

Lo scenario

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

Questo esempio mostra come configurare ed eseguire un'app Web che presuppone un ruolo IAM per la lettura da un bucket Amazon S3. L'esempio utilizza la libreria front-end React e gli strumenti front-end Vite per fornire un ambiente di sviluppo. JavaScript L'app Web utilizza un pool di identità Amazon Cognito per fornire le credenziali necessarie per accedere ai servizi. AWS L'esempio di codice incluso illustra i modelli di base per il caricamento e l'utilizzo dell' AWS 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 Console di gestione AWS 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 Configure identity pool trust, scegli Accesso ospite per l'autenticazione degli utenti.
5. In Configura le autorizzazioni, scegli Crea un nuovo ruolo IAM e inserisci un nome (ad esempio, getStartedRole) nel nome del ruolo IAM.

6. In Configure properties, inserisci un nome (ad esempio, getStartedPool) in Identity pool name.
7. In Esamina e crea, conferma le selezioni effettuate per il nuovo pool di identità. Seleziona Modifica per tornare alla procedura guidata e modificare le eventuali impostazioni. Al termine, seleziona Crea un pool di identità.
8. Prendi nota dell'ID del pool di identità e della regione del pool di identità di Amazon Cognito appena creato. Questi valori sono necessari per sostituirli *IDENTITY_POOL_ID* e *REGION* inserirli. Passaggio 4: configura il codice del browser

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

Passaggio 2: aggiungi una policy al ruolo IAM creato

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

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

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

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

6. Scegli Add Permissions (Aggiungi autorizzazioni).

Dopo aver creato il tuo pool di identità Amazon Cognito e 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 a Console di gestione AWS 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. Scegli Save changes (Salva 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:

   - *`YOUR_IDENTITY_POOL_ID`*Sostituiscilo con l'ID del pool di identità di Amazon Cognito in cui hai registrato. 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).
   - *`bucket-name`*Sostituiscilo 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 `App.tsx` file. 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.

Pulizia

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

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

Guida introduttiva a React Native

Questo tutorial mostra come creare un'app React Native utilizzando la CLI React Native.

Questo tutorial ti mostra:

- Come installare e includere l' AWS SDK per i moduli della JavaScript versione 3 (V3) utilizzati dal progetto.
- Come scrivere codice che si connetta ad Amazon Simple Storage Service (Amazon S3) per creare ed eliminare un bucket Amazon S3.

Lo scenario

Amazon S3 è un servizio cloud che consente di archiviare e recuperare qualsiasi quantità di dati in qualsiasi momento, da qualsiasi punto del Web. React Native è un framework di sviluppo che consente di creare applicazioni mobili. Questo tutorial mostra come creare un'app React Native che si connette ad Amazon S3 per creare ed eliminare un bucket Amazon S3.

L'app utilizza il seguente AWS SDK per: JavaScript APIs

- Costruttore `CognitoIdentityClient`
- Costruttore `S3`

Attività prerequisite

> **Note**
>
> Se hai già completato uno dei passaggi seguenti con altri tutorial o configurazioni, ignora questi passaggi.

Questa sezione fornisce la configurazione minima necessaria per completare questo tutorial. Non la considerare come una configurazione completa. Per la configurazione completa, vedi Configura l'SDK per JavaScript.

- Installa i seguenti strumenti:
  - npm
  - Node.js
  - Xcode se stai testando su iOS
  - Android Studio se esegui il test su Android

- Configura il tuo ambiente di sviluppo React Native
- Configura l'ambiente di progetto per eseguire questi TypeScript esempi di Node e installa l' AWS SDK richiesto per i JavaScript moduli di terze parti. Segui le istruzioni su. GitHub
- È necessario stabilire in che modo il codice si autentica AWS durante lo sviluppo con AWS i servizi. Per ulteriori informazioni, consulta Autenticazione SDK con AWS.

> **Note**
>
> Il ruolo IAM per questo esempio deve essere impostato per utilizzare le autorizzazioni FullAccessAmazonS3.

Fase 1: creare un pool di identità di Amazon Cognito

In questo esercizio, crei e utilizzi un pool di identità Amazon Cognito per fornire un accesso non autenticato alla tua app per il servizio Amazon S3. La creazione di un pool di identità crea anche due ruoli AWS Identity and Access Management (IAM), uno per supportare gli utenti autenticati da un provider di identità e l'altro per supportare gli utenti guest non autenticati.

In questo esercizio, lavoreremo solo con il ruolo degli utenti non autenticati in modo che l'attività risulti mirata. È possibile integrare il supporto per un provider di identità e per gli utenti autenticati in un secondo momento.

Per creare un pool di identità Amazon Cognito

1. Accedi Console di gestione AWS e apri la console Amazon Cognito su Amazon Web Services Console.
2. Scegli Identity Pools nella pagina di apertura della console.
3. Nella pagina successiva, scegliere Create new identity pool (Crea nuovo pool di identità).

   > **Note**
   >
   > Se non sono presenti altri pool di identità, la console Amazon Cognito salterà questa pagina e aprirà invece la pagina successiva.

4. In Configure identity pool trust, 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, getStartedReactRuolo) nel nome del ruolo IAM.

6. In Configure properties, inserisci un nome (ad esempio, getStartedReactPool) nel campo Identity pool name.
7. In Esamina e crea, conferma le selezioni effettuate per il nuovo pool di identità. Seleziona Modifica per tornare alla procedura guidata e modificare le eventuali impostazioni. Al termine, seleziona Crea un pool di identità.
8. Annota l'ID del pool di identità e la regione per questo pool di identità appena creato. Questi valori devono essere sostituiti *`region`* e inseriti *`identityPoolId`* nello script 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 React Native.

Fase 2: aggiungere una policy al ruolo IAM creato

Per consentire l'accesso tramite script del browser ad Amazon S3 per creare ed eliminare un bucket Amazon S3, utilizza il ruolo IAM non autenticato creato per il tuo pool di identità Amazon Cognito. Ciò richiede l'aggiunta di una policy IAM al ruolo. Per ulteriori informazioni sui ruoli IAM, consulta Creating a Role to Delegate Permissions to an AWS Service nella IAM User Guide.

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

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

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

6. Scegli Add Permissions (Aggiungi autorizzazioni).

Dopo aver creato il tuo pool di identità Amazon Cognito e aggiunto le autorizzazioni per Amazon S3 al tuo ruolo IAM per utenti non autenticati, sei pronto per creare l'app.

Passaggio 3: crea l'app utilizzando create-react-native-app

Crea un'app React Native eseguendo il seguente comando.

```
npx react-native init ReactNativeApp --npm
```

Fase 4: Installare il pacchetto Amazon S3 e altre dipendenze

All'interno della directory del progetto, esegui i seguenti comandi per installare il pacchetto Amazon S3.

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

Questo comando installa il pacchetto Amazon S3 nel progetto e lo `package.json` aggiorna per elencare Amazon S3 come dipendenza del progetto. Puoi trovare informazioni su questo pacchetto cercando "@aws -sdk» sul sito web di npm. https://www.npmjs.com/

Questi pacchetti e il relativo codice vengono installati nella sottodirectory `node_modules` del progetto.

Per ulteriori informazioni sull'installazione dei pacchetti Node.js, consulta Scaricamento e installazione dei pacchetti localmente e Creazione dei moduli Node.js sul sito web di npm (Node.js package manager). Per informazioni sul download e l'installazione dell' AWS SDK per JavaScript, consulta. Installa l'SDK per JavaScript

Installa le altre dipendenze necessarie per l'autenticazione.

```
npm install @aws-sdk/client-cognito-identity @aws-sdk/credential-provider-cognito-
identity
```

Passaggio 5: scrivere il codice React Native

Aggiungi il codice seguente a`App.tsx`. Sostituisci *`identityPoolId`* e *`region`* con l'ID del pool di identità e la regione in cui verrà creato il bucket Amazon S3.

```
import React, { useCallback, useState } from "react";
import { Button, StyleSheet, Text, TextInput, View } from "react-native";
```

```
import "react-native-get-random-values";
import "react-native-url-polyfill/auto";

import {
  S3Client,
  CreateBucketCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";

const client = new S3Client({
  // The AWS Region where the Amazon Simple Storage Service (Amazon S3) bucket will be
 created. Replace this with your Region.
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    // Replace the value of 'identityPoolId' with the ID of an Amazon Cognito identity
 pool in your Amazon Cognito Region.
    identityPoolId: "us-east-1:edbe2c04-7f5d-469b-85e5-98096bd75492",
    // Replace the value of 'region' with your Amazon Cognito Region.
    clientConfig: { region: "us-east-1" },
  }),
});

enum MessageType {
  SUCCESS = 0,
  FAILURE = 1,
  EMPTY = 2,
}

const App = () => {
  const [bucketName, setBucketName] = useState("");
  const [msg, setMsg] = useState<{ message: string; type: MessageType }>({
    message: "",
    type: MessageType.EMPTY,
  });

  const createBucket = useCallback(async () => {
    setMsg({ message: "", type: MessageType.EMPTY });

    try {
      await client.send(new CreateBucketCommand({ Bucket: bucketName }));
      setMsg({
        message: `Bucket "${bucketName}" created.`,
        type: MessageType.SUCCESS,
```

```
      });
    } catch (e) {
      console.error(e);
      setMsg({
        message: e instanceof Error ? e.message : "Unknown error",
        type: MessageType.FAILURE,
      });
    }
  }, [bucketName]);

  const deleteBucket = useCallback(async () => {
    setMsg({ message: "", type: MessageType.EMPTY });

    try {
      await client.send(new DeleteBucketCommand({ Bucket: bucketName }));
      setMsg({
        message: `Bucket "${bucketName}" deleted.`,
        type: MessageType.SUCCESS,
      });
    } catch (e) {
      setMsg({
        message: e instanceof Error ? e.message : "Unknown error",
        type: MessageType.FAILURE,
      });
    }
  }, [bucketName]);

  return (
    <View style={styles.container}>
      {msg.type !== MessageType.EMPTY && (
        <Text
          style={
            msg.type === MessageType.SUCCESS
              ? styles.successText
              : styles.failureText
          }
        >
          {msg.message}
        </Text>
      )}
      <View>
        <TextInput
          onChangeText={(text) => setBucketName(text)}
          autoCapitalize={"none"}
```

```
          value={bucketName}
          placeholder={"Enter Bucket Name"}
        />
        <Button color="#68a0cf" title="Create Bucket" onPress={createBucket} />
        <Button color="#68a0cf" title="Delete Bucket" onPress={deleteBucket} />
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: "center",
    justifyContent: "center",
  },
  successText: {
    color: "green",
  },
  failureText: {
    color: "red",
  },
});

export default App;
```

Le prime importazioni di codice richiedevano le dipendenze React, React Native e AWS SDK.

All'interno della funzione App:

- L'oggetto S3Client viene creato, specificando le credenziali utilizzando Amazon Cognito Identity Pool creato in precedenza.
- I metodi `deleteBucket` creano `createBucket` ed eliminano rispettivamente il bucket specificato.
- React Native View visualizza un campo di immissione di testo per consentire all'utente di specificare il nome di un bucket Amazon S3 e pulsanti per creare ed eliminare il bucket Amazon S3 specificato.

La JavaScript pagina completa è disponibile qui. GitHub

Fase 6: Esegui l'esempio

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

Per eseguire l'esempio `webios`, esegui o `android` comanda utilizzando npm.

Ecco un esempio di output del `ios` comando in esecuzione su macOS.

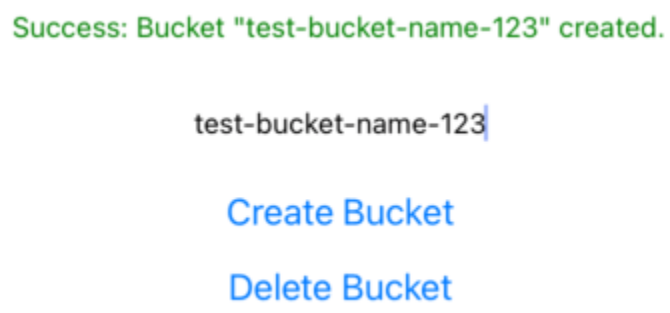
```
$ npm run ios

> ReactNativeApp@0.0.1 ios /Users/trivikr/workspace/ReactNativeApp
> react-native run-ios

info Found Xcode workspace "ReactNativeApp.xcworkspace"
info Launching iPhone 11 (iOS 14.2)
info Building (using "xcodebuild -workspace ReactNativeApp.xcworkspace -configuration
 Debug -scheme ReactNativeApp -destination id=706C1A97-FA38-407D-AD77-CB4FCA9134E9")
success Successfully built the app
info Installing "/Users/trivikr/Library/Developer/Xcode/DerivedData/ReactNativeApp-
cfhmsyhptwflqqejyspdqgjestra/Build/Products/Debug-iphonesimulator/ReactNativeApp.app"
info Launching "org.reactjs.native.example.ReactNativeApp"

success Successfully launched the app on the simulator
```

Ecco un esempio di output del `android` comando in esecuzione su macOS.

```
$ npm run android

> ReactNativeApp@0.0.1 android
> react-native run-android

info Running jetifier to migrate libraries to AndroidX. You can disable it using "--no-
jetifier" flag.
Jetifier found 970 file(s) to forward-jetify. Using 12 workers...
info Starting JS server...
info Launching emulator...
info Successfully launched emulator.
```

```
info Installing the app...

> Task :app:stripDebugDebugSymbols UP-TO-DATE
Compatible side by side NDK version was not found.

> Task :app:installDebug
02:18:38 V/ddms: execute: running am get-config
02:18:38 V/ddms: execute 'am get-config' on 'emulator-5554' : EOF hit. Read: -1
02:18:38 V/ddms: execute: returning
Installing APK 'app-debug.apk' on 'Pixel_3a_API_30_x86(AVD) - 11' for app:debug
02:18:38 D/app-debug.apk: Uploading app-debug.apk onto device 'emulator-5554'
02:18:38 D/Device: Uploading file onto device 'emulator-5554'
02:18:38 D/ddms: Reading file permission of /Users/trivikr/workspace/ReactNativeApp/
android/app/build/outputs/apk/debug/app-debug.apk as: rw-r--r--
02:18:40 V/ddms: execute: running pm install -r -t "/data/local/tmp/app-debug.apk"
02:18:41 V/ddms: execute 'pm install -r -t "/data/local/tmp/app-debug.apk"' on
 'emulator-5554' : EOF hit. Read: -1
02:18:41 V/ddms: execute: returning
02:18:41 V/ddms: execute: running rm "/data/local/tmp/app-debug.apk"
02:18:41 V/ddms: execute 'rm "/data/local/tmp/app-debug.apk"' on 'emulator-5554' : EOF
 hit. Read: -1
02:18:41 V/ddms: execute: returning
Installed on 1 device.

Deprecated Gradle features were used in this build, making it incompatible with Gradle
 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.2/userguide/
command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 6s
27 actionable tasks: 2 executed, 25 up-to-date
info Connecting to the development server...
8081
info Starting the app on "emulator-5554"...
Starting: Intent { cmp=com.reactnativeapp/.MainActivity }
```

Inserisci il nome del bucket che desideri creare o eliminare e fai clic su Crea bucket o Elimina bucket. Il rispettivo comando verrà inviato ad Amazon S3 e verrà visualizzato un messaggio di successo o di errore.



Possibili miglioramenti

Ecco alcune varianti di questa applicazione che puoi utilizzare per esplorare ulteriormente l'utilizzo dell' AWS SDK JavaScript in un'app React Native.

- Aggiungi un pulsante per elencare i bucket Amazon S3 e fornisci un pulsante di eliminazione accanto a ciascun bucket elencato.
- Aggiungi un pulsante per inserire un oggetto di testo in un bucket.
- Integra un provider di identità esterno come Facebook o Amazon da utilizzare con il ruolo IAM autenticato.

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.

Argomenti

Prerequisiti

Installa Node.js. AWS consiglia di utilizzare la versione Active LTS di Node.js per lo sviluppo.

Argomenti

Configurare 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' EC2 istanza Amazon utilizzando quell'AMI. Quando crei la tua EC2 istanza Amazon, 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' EC2 istanza Amazon 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' EC2 istanza 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

Browser Web supportati

AWS SDK for JavaScript Supporta tutti i browser Web moderni.

Nella versione 3.567.0 o successiva, l'SDK JavaScript emette ES2 021 artefatti, che supportano le seguenti versioni minime.

Browser	Versione
Google Chrome	85,0 o più
Mozilla Firefox	80,0+
Opera	71,0 +
Microsoft Edge	85,0+
Apple Safari	14,1+
Samsung Internet	14,0+

Nella versione da 3.183.0 a 3.566.0, l'SDK JavaScript utilizza ES2 020 artefatti, che supportano le seguenti versioni minime.

Browser	Versione
Google Chrome	80,0 o versioni successive
Mozilla Firefox	80,0+
Opera	63,0+
Microsoft Edge	80,0+
Apple Safari	14,1+

Browser	Versione
Samsung Internet	12,0+

Nella versione 3.182.0 o precedente, l'SDK for JavaScript utilizza ES5 artefatti, che supporta 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, *SERVICE* dov'è il nome di un servizio, ad esempio. `s3`

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

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

Carica l'SDK per JavaScript

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

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

Configurare l'SDK per JavaScript

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

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

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

La AWS SDKs and 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 l'SDK JavaScript per Node.js e come JavaScript eseguirlo in un browser Web.

Argomenti

Configurazione per servizio

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

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

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

Imposta la configurazione per servizio

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

Ad esempio, se devi accedere 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). Quando si crea un client di servizio nell'SDK, si specifica una regione in JavaScript modo che l'SDK acceda al servizio in quella regione. Alcuni servizi sono disponibili solo in regioni specifiche.

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

In un costruttore di classi client

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

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

Usa una variabile di ambiente

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

Usa un file di configurazione condiviso

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

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

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

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

Per ulteriori informazioni sull'utilizzo di `credentials` file `config` e file condivisi, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli 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 l'API. Le credenziali possono essere impostate per servizio, passando le credenziali direttamente a un oggetto di servizio.

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

Procedure consigliate per le credenziali

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

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

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

Argomenti

Impostare le credenziali in Node.js

Consigliamo ai nuovi utenti che stanno sviluppando localmente e che non dispongono di un metodo di autenticazione dal datore di lavoro di effettuare la configurazione AWS IAM Identity Center. 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 SDKs hanno una serie di luoghi (o fonti) che controllano per ottenere credenziali valide da utilizzare per effettuare una richiesta a un. Servizio AWS Dopo aver trovato credenziali valide, la ricerca viene interrotta. Questa ricerca sistematica è chiamata catena di fornitori di credenziali predefinita.

Per ogni fase della catena, esistono diversi modi per impostare i valori. L'impostazione dei valori direttamente nel codice ha sempre la precedenza, seguita dall'impostazione come variabili di ambiente e quindi nel AWS `config` file condiviso. Per ulteriori informazioni, vedete la precedenza delle impostazioni nella AWS SDKs and Tools Reference Guide.

La AWS SDKs and Tools Reference Guide contiene informazioni sulle impostazioni di configurazione SDK utilizzate da tutti AWS SDKs 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 JavaScript API Reference: metodo del provider di credenziali in base alla precedenza	Provider di credenziali disponibili	AWS SDKs e guida di riferimento agli strumenti
`fromEnv()`	AWS chiavi di accesso dalle variabili di ambiente	AWS chiavi di accesso
`fromSSO()`	AWS IAM Identity Center. In questa guida, vediAutenticazione 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 di istanza 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 JavaScript API Reference: metodo del provider di credenziali in base alla precedenza	Provider di credenziali disponibili	AWS SDKs e guida di riferimento agli strumenti
	AWS IAM Identity Center	Fornitore di credenziali IAM Identity Center
	Provider di credenziali di accesso	Fornitore di credenziali di accesso
fromLoginCredentials()	Fornitore di credenziali di accesso	Fornitore di credenziali di accesso
fromProcess()	Fornitore 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
fromContainerMetadata()	Credenziali Amazon Elastic Container Service (Amazon ECS)	Fornitore di credenziali per container
fromInstanceMetadata()	Credenziali del profilo di istanza 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 strumenti AWS SDKs 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

Se esegui l'applicazione Node.js su un' EC2 istanza Amazon, puoi sfruttare i ruoli IAM per Amazon per EC2 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' EC2 istanza Amazon, 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

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

> **Note**
>
> AWS consiglia di utilizzare la versione Active LTS di Node.js per lo sviluppo.

Utilizza i moduli Node.js integrati

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

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

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

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

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

Usa i pacchetti npm

Oltre ai moduli integrati, puoi anche includere e incorporare codice di terze parti proveniente dal `npm` gestore di pacchetti Node.js. Si tratta di un repository di pacchetti Node.js open source e di un'interfaccia a riga di comando per installare i pacchetti. Per ulteriori informazioni `npm` e per un elenco dei pacchetti attualmente disponibili, vedere https://www.npmjs.com. Puoi anche saperne di più sui pacchetti Node.js aggiuntivi che puoi usare qui GitHub.

Configurare MaxSockets in Node.js

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

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

L'esempio seguente mostra come impostare `maxSockets` un client DynamoDB.

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

```
});

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

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

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

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

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

Riutilizza le connessioni con keep-alive in Node.js

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

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

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

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "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, l'SDK JavaScript supporta l'uso di proxy HTTP o HTTPS tramite un agente HTTP di terze parti.

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

Per installare un agente proxy HTTP di terze parti, inserisci quanto segue al prompt dei comandi, *`PROXY`* dov'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 per l'utilizzo degli script nei browser. AWS SDK for JavaScript

Argomenti

- Crea l'SDK per i browser
- Cross-Origin Resource Sharing (CORS)

- Raggruppa le applicazioni con webpack

Crea l'SDK per i browser

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

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

Usa SDK Builder per creare l'SDK per JavaScript

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

Cross-Origin Resource Sharing (CORS)

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

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

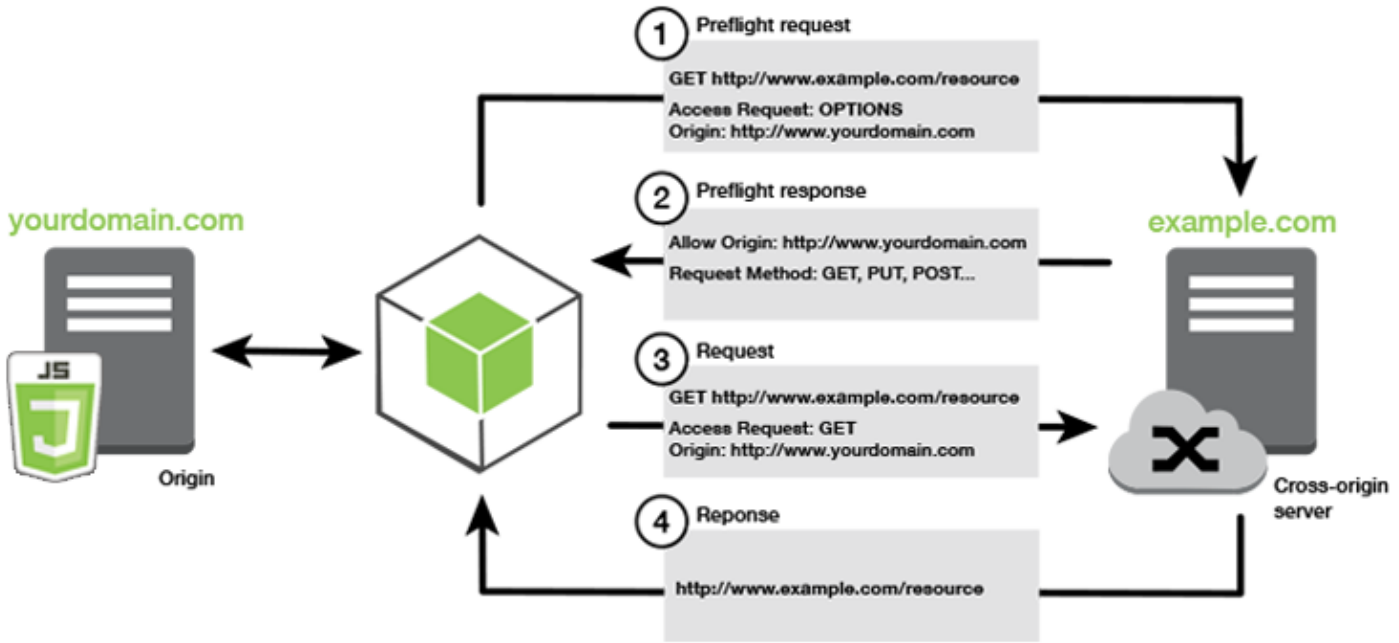
CORS determina se consentire la condivisione di risorse in una richiesta tra origini diverse in base a quanto segue:

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

Come funziona CORS

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

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



È richiesta la configurazione CORS?

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

Configurare CORS per un bucket Amazon S3

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

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

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

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

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

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

- `<AllowedHeader>`— Speciifica le intestazioni consentite in una richiesta di preflight.

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

Esempio di configurazione CORS

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

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

XML

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

JSON

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

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

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

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

Raggruppa le applicazioni con webpack

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

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

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

Installa webpack

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

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

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

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

Configura webpack

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

> **Note**
>
> I requisiti di configurazione di Webpack variano a seconda della versione di Webpack installata. Per ulteriori informazioni, consulta la documentazione di Webpack.

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

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

In questo esempio, `browser.js` viene specificato come punto di ingresso. Il punto di ingresso è il file `webpack` utilizzato per iniziare la ricerca dei moduli importati. Come `bundle.js` è specificato il nome del file di output. Questo file di output conterrà tutto JavaScript il necessario per l'esecuzione dell'applicazione. Se il codice specificato nel punto di ingresso importa o richiede altri moduli, come l'SDK for JavaScript, quel codice viene fornito in bundle senza bisogno di specificarlo nella configurazione.

Esegui webpack

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

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

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

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

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

Per creare la tua applicazione, inserisci il seguente comando.

```
npm run build
```

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

Usa il pacchetto webpack

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

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

Bundle per Node.js

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

```
target: "node"
```

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

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

Lavora con AWS i servizi nell'SDK per JavaScript

La AWS SDK for JavaScript versione v3 fornisce l'accesso ai servizi che supporta tramite una raccolta di classi client. Da queste classi client, si creano oggetti di interfaccia di servizio, comunemente chiamati oggetti di servizio. Ogni AWS servizio supportato include una o più classi 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 l'SDK JavaScript seguono lo schema di richiesta-risposta per lo scambio di messaggi con le applicazioni chiamanti. In questo modello, il codice che richiama un servizio inoltra una richiesta HTTP/HTTPS a un endpoint per il servizio. La richiesta contiene i parametri necessari per richiamare correttamente la funzionalità specifica chiamata. Il servizio richiamato genera una risposta che viene inviata nuovamente al richiedente. La risposta contiene dati se l'operazione ha avuto esito positivo o informazioni di errore se l'operazione non ha avuto esito positivo.

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

Argomenti

Crea e richiama oggetti di servizio

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

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

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

Specificare i parametri dell'oggetto di servizio

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

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

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

Usa @smithy /types per i client generati in TypeScript

Se lo stai utilizzando TypeScript, il `@smithy/types` pacchetto ti consente di manipolare le forme di input e output di un client.

Scenario: rimozione **`undefined`** dalle strutture di input e output

I membri delle forme generate vengono uniti alle forme `undefined` di input e `?` (facoltativo) alle forme di output. Per gli input, ciò rinvia la convalida al servizio. Per quanto riguarda gli output, ciò suggerisce vivamente di controllare in fase di esecuzione i dati di output.

Se desideri saltare questi passaggi, usa gli helper o digitaAssertiveClient. UncheckedClient L'esempio seguente utilizza i type helper con il servizio Amazon S3.

```
import { S3 } from "@aws-sdk/client-s3";
import type { AssertiveClient, UncheckedClient } from "@smithy/types";

const s3a = new S3({}) as AssertiveClient<S3>;
const s3b = new S3({}) as UncheckedClient<S3>;

// AssertiveClient enforces required inputs are not undefined
// and required outputs are not undefined.
const get = await s3a.getObject({
  Bucket: "",
  // @ts-expect-error (undefined not assignable to string)
  Key: undefined,
});

// UncheckedClient makes output fields non-nullable.
// You should still perform type checks as you deem
// necessary, but the SDK will no longer prompt you
// with nullability errors.
const body = await (
  await s3b.getObject({
    Bucket: "",
    Key: "",
  })
).Body.transformToString();
```

Quando si utilizza la trasformazione su un client non aggregato con la Command sintassi, l'input non può essere convalidato perché passa attraverso un'altra classe, come mostrato nell'esempio seguente.

```
import { S3Client, ListBucketsCommand, GetObjectCommand, GetObjectCommandInput } from
 "@aws-sdk/client-s3";
import type { AssertiveClient, UncheckedClient, NoUndefined } from "@smithy/types";

const s3 = new S3Client({}) as UncheckedClient<S3Client>;

const list = await s3.send(
  new ListBucketsCommand({
    // command inputs are not validated by the type transform.
    // because this is a separate class.
```

```
  })
);

/**
 * Although less ergonomic, you can use the NoUndefined<T>
 * transform on the input type.
 */
const getObjectInput: NoUndefined<GetObjectCommandInput> = {
  Bucket: "undefined",
  // @ts-expect-error (undefined not assignable to string)
  Key: undefined,
  // optional params can still be undefined.
  SSECustomerAlgorithm: undefined,
};

const get = s3.send(new GetObjectCommand(getObjectInput));

// outputs are still transformed.
await get.Body.TransformToString();
```

Scenario: restringimento dei tipi di blob del payload di output di un client generato da Smithy TypeScript

Questo scenario è principalmente rilevante per le operazioni con sistemi di streaming, ad esempio all'interno di in v3. `S3Client` AWS SDK for JavaScript

Poiché i tipi di payload blob dipendono dalla piattaforma, potresti voler indicare nell'applicazione che un client è in esecuzione in un ambiente specifico. Questo restringe i tipi di payload blob, come illustrato nell'esempio seguente.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import type { NodeJsClient, SdkStream, StreamingBlobPayloadOutputTypes } from "@smithy/
types";
import type { IncomingMessage } from "node:http";

// default client init.
const s3Default = new S3Client({});

// client init with type narrowing.
const s3NarrowType = new S3Client({}) as NodeJsClient<S3Client>;

// The default type of blob payloads is a wide union type including multiple possible
```

```
// request handlers.
const body1: StreamingBlobPayloadOutputTypes = (await s3Default.send(new
 GetObjectCommand({ Key: "", Bucket: "" })))
  .Body!;

// This is of the narrower type SdkStream<IncomingMessage> representing
// blob payload responses using specifically the node:http request handler.
const body2: SdkStream<IncomingMessage> = (await s3NarrowType.send(new
 GetObjectCommand({ Key: "", Bucket: "" })))
  .Body!;
```

Chiama i servizi in modo asincrono

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

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

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

async/await

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

promettere

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

richiamata

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

Argomenti

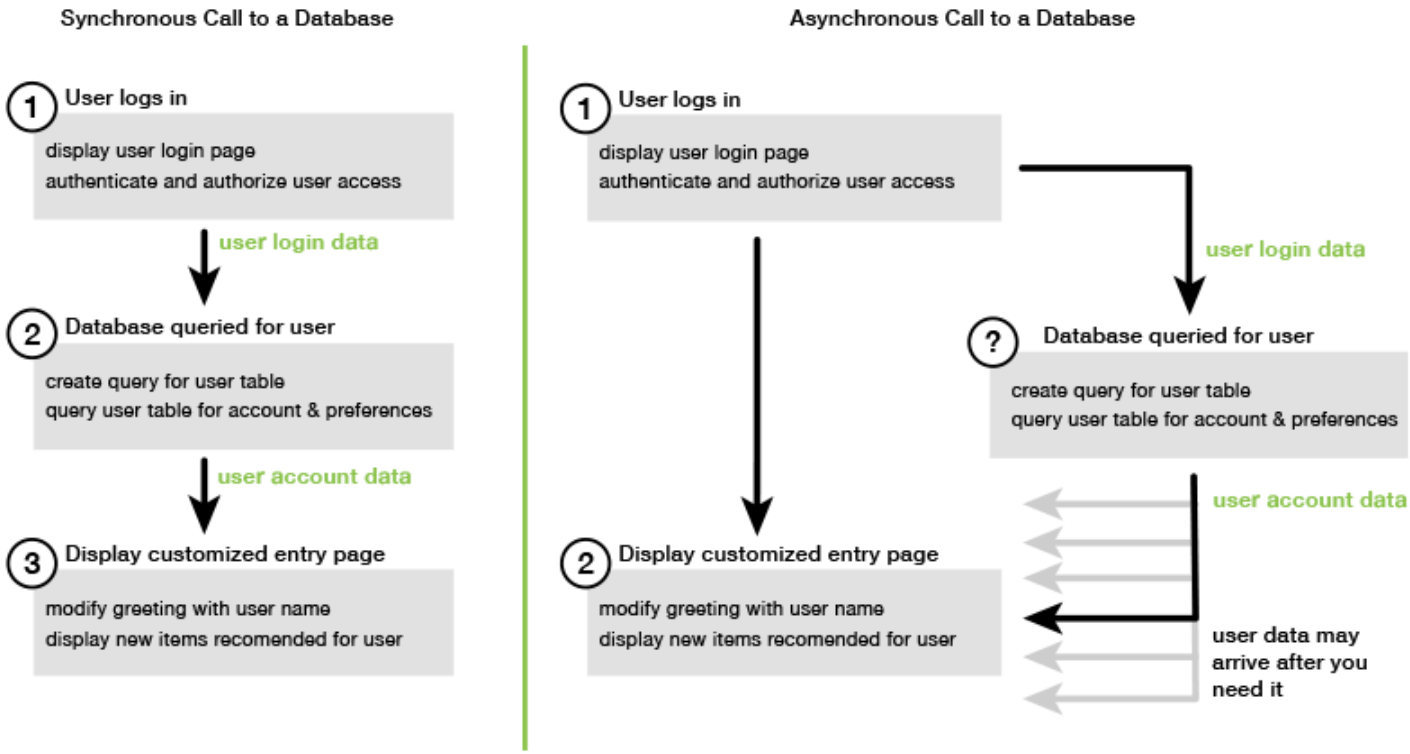
Gestisci le chiamate asincrone

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

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

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

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



Usa async/await

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

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

> **Note**
>
> Per eseguire questo esempio:
>
> - Installa il client AWS SDK for JavaScript DynamoDB `npm install @aws-sdk/client-dynamodb` entrando nella 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 (ECMAScript 2015) dipende dal JavaScript motore e dalla versione in cui viene eseguito il codice. Per determinare il supporto per le JavaScript promesse in ogni ambiente in cui il codice deve essere eseguito, consulta la tabella di 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) dell'SDK JavaScript consente di inviare richieste.

> **Note**
>
> È inoltre possibile eseguire operazioni utilizzando i comandi della versione 2 (V2) quando si utilizza la versione 3 dell'SDK for. JavaScript Per ulteriori informazioni, consulta Utilizzo dei comandi v2.

Per inviare una richiesta:

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

Ad esempio, per elencare le tue tabelle Amazon DynamoDB, puoi farlo con `us-west-2` async/await.

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

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

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

Gestisci le risposte dei clienti di assistenza

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

Accedi ai dati restituiti nella risposta

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

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

Informazioni sugli errori di accesso

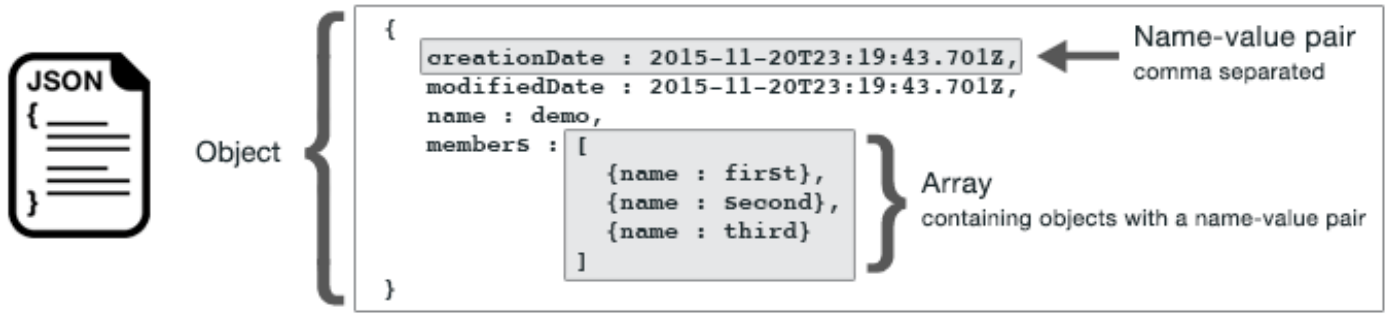
Se un comando fallisce, genera un'eccezione. Il seguente frammento di codice mostra un modo per gestire un'eccezione di servizio.

```
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
  }
}
```

Lavora con JSON

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

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



JSON rappresenta i dati in due modi:

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

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

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

JSON come parametri dell'oggetto di servizio

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

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

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

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

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

Registrazione delle chiamate AWS SDK for JavaScript

AWS SDK for JavaScript È dotato di un logger integrato che consente di registrare le chiamate API effettuate con l'SDK per. JavaScript

Per attivare il logger e stampare le voci di registro nella console, configura il client di servizio utilizzando il parametro opzionale. `logger` L'esempio seguente abilita la registrazione del client ignorando gli output di trace e debug.

```
new S3Client({
  logger: {
    ...console,
    debug(...args) {},
    trace(...args) {},
  },
});
```

Utilizzo del middleware per registrare le richieste

AWS SDK for JavaScript Utilizza uno stack middleware per controllare il ciclo di vita di una chiamata operativa. Ogni middleware dello stack chiama il middleware successivo dopo aver apportato modifiche all'oggetto della richiesta. Inoltre, ciò semplifica notevolmente i problemi di debug nello

stack, poiché è possibile vedere esattamente quale middleware è stato chiamato a causare un errore. Ecco un esempio di registrazione delle richieste tramite middleware:

```
const client = new DynamoDB({ region: "us-west-2" });

client.middlewareStack.add(
  (next, context) => async (args) => {
    console.log("AWS SDK context", context.clientName, context.commandName);
    console.log("AWS SDK request input", args.input);
    const result = await next(args);
    console.log("AWS SDK request output:", result.output);
    return result;
  },
  {
    name: "MyMiddleware",
    step: "build",
    override: true,
  }
);

await client.listTables({});
```

Nell'esempio precedente, un middleware viene aggiunto allo stack middleware del client DynamoDB. Il primo argomento è una funzione che accetta`next`, il middleware successivo nello stack da chiamare e un oggetto che contiene alcune informazioni sull'`context`operazione chiamata. Restituisce una funzione che accetta`args`, un oggetto che contiene i parametri passati all'operazione e alla richiesta e restituisce il risultato della chiamata al middleware successivo con. `args`

Usa endpoint AWS basati su account con DynamoDB

DynamoDB AWS offre endpoint basati su account che possono migliorare le prestazioni utilizzando l'ID dell'account per semplificare AWS il routing delle richieste.

Per sfruttare questa funzionalità, è necessario utilizzare la versione 3.656.0 o successiva della versione 3. AWS SDK for JavaScript Questa funzionalità degli endpoint basata sull'account è abilitata per impostazione predefinita in questa nuova versione.

Se desideri disattivare il routing basato sull'account, hai le seguenti opzioni:

- Configurare un client di servizio DynamoDB con `accountIdEndpointMode` il parametro impostato su. `disabled`

- Imposta la variabile `AWS_ACCOUNT_ID_ENDPOINT_MODE` di ambiente su. `disabled`
- Aggiorna l'impostazione del file di AWS configurazione condiviso `account_id_endpoint_mode` su`disabled`.

Il seguente frammento è un esempio di come disabilitare il routing basato su account configurando un client di servizio DynamoDB:

```
const ddbClient = new DynamoDBClient({
  region: "us-west-2",
  accountIdEndpointMode: "disabled" // Disable account ID in the endpoint
});
```

La AWS SDKs and Tools Reference Guide fornisce ulteriori informazioni sulle altre opzioni di configurazione.

Protezione dell'integrità dei dati con i checksum di Amazon S3

Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) offre la possibilità di specificare un checksum quando carichi un oggetto. Quando specifichi un checksum, questo viene memorizzato con l'oggetto e può essere convalidato quando l'oggetto viene scaricato.

I checksum forniscono un ulteriore livello di integrità dei dati durante il trasferimento dei file. Con i checksum, è possibile verificare la coerenza dei dati confermando che il file ricevuto corrisponde al file originale. Per ulteriori informazioni sui checksum con Amazon S3, consulta la Guida per l'utente di Amazon Simple Storage Service, inclusi gli algoritmi supportati.

Hai la flessibilità di scegliere l'algoritmo più adatto alle tue esigenze e lasciare che sia l'SDK a calcolare il checksum. In alternativa, puoi fornire un valore di checksum precalcolato utilizzando uno degli algoritmi supportati.

> **Note**
>
> A partire dalla versione 3.729.0 di AWS SDK for JavaScript, l'SDK fornisce protezioni di integrità predefinite calcolando automaticamente un checksum per i caricamenti. `CRC32` L'SDK calcola questo checksum se non fornisci un valore di checksum precalcolato o se non specifichi un algoritmo che l'SDK deve utilizzare per calcolare un checksum.

L'SDK fornisce anche impostazioni globali per la protezione dell'integrità dei dati che puoi impostare esternamente, come puoi leggere nella Guida di riferimento agli strumenti e agli strumenti.AWS SDKs

Caricamento di un oggetto

Carichi oggetti su Amazon S3 utilizzando il PutObjectcomando di. `S3Client` Utilizza il `ChecksumAlgorithm` parametro del generatore per abilitare il calcolo del `PutObjectRequest` checksum e specificare l'algoritmo. Consulta gli algoritmi di checksum supportati per i valori validi.

Il seguente frammento di codice mostra una richiesta di caricamento di un oggetto con un checksum CRC-32. Quando l'SDK invia la richiesta, calcola il checksum CRC-32 e carica l'oggetto. Amazon S3 memorizza il checksum con l'oggetto.

```
import { ChecksumAlgorithm, S3 } from "@aws-sdk/client-s3";

const client = new S3();
const response = await client.putObject({
  Bucket: "my-bucket",
  Key: "my-key",
  Body: "Hello, world!",
  ChecksumAlgorithm: ChecksumAlgorithm.CRC32,
});
```

Se non fornisci un algoritmo di checksum con la richiesta, il comportamento del checksum varia a seconda della versione dell'SDK che usi, come mostrato nella tabella seguente.

Comportamento del checksum quando non viene fornito alcun algoritmo di checksum

SDK per versione JavaScript	Comportamento del checksum
Precedente alla 3.729.0	L'SDK non calcola automaticamente un checksum basato su CRC e lo fornisce nella richiesta.
3.729.0 o versione successiva	L'SDK utilizza l'`CRC32`algoritmo per calcolare il checksum e lo fornisce nella richiesta. Amazon S3 convalida l'integrità del trasferimento

SDK per versione JavaScript	Comportamento del checksum
	calcolando il proprio `CRC32` checksum e lo confronta con il checksum fornito dall'SDK. Se i checksum corrispondono, il checksum viene salvato con l'oggetto.

Se il checksum calcolato dall'SDK non corrisponde al checksum calcolato da Amazon S3 quando riceve la richiesta, viene restituito un errore.

Utilizza un valore di checksum precalcolato

Un valore di checksum precalcolato fornito con la richiesta disabilita il calcolo automatico da parte dell'SDK e utilizza invece il valore fornito.

L'esempio seguente mostra una richiesta con un checksum SHA-256 precalcolato.

```
import { S3 } from "@aws-sdk/client-s3";
import { createHash } from "node:crypto";

const client = new S3();

const Body = "Hello, world!";
const ChecksumSHA256 = await createHash("sha256").update(Body).digest("base64");

const response = await client.putObject({
  Bucket: "my-bucket",
  Key: "my-key",
  Body,
  ChecksumSHA256,
});
```

Se Amazon S3 determina che il valore del checksum non è corretto per l'algoritmo specificato, il servizio restituisce una risposta di errore.

Caricamenti in più parti

Puoi anche utilizzare i checksum con caricamenti in più parti. AWS SDK for JavaScript Possono utilizzare le opzioni della `Upload` libreria to from per utilizzare i checksum con `@aws-sdk/lib-storage` caricamenti in più parti.

```
import { ChecksumAlgorithm, S3 } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";
import { createReadStream } from "node:fs";

const client = new S3();
const filePath = "/path/to/file";
const Body = createReadStream(filePath);

const upload = new Upload({
  client,
  params: {
    Bucket: "my-bucket",
    Key: "my-key",
    Body,
    ChecksumAlgorithm: ChecksumAlgorithm.CRC32,
  },
});
await upload.done();
```

SDK per 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 rispetto ai semplici frammenti di codice che coprono solo le singole chiamate API. Per istruzioni, consultate la sezione Codice di creazione nelle linee guida per i contributi su GitHub.

Important

Questi esempi utilizzano la sintassi di 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 ES6Sintassi /CommonJS per la conversione.

Argomenti

JavaScript ES6Sintassi /CommonJS

Gli esempi di AWS 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 della 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, puoi convertire uno qualsiasi dei nostri esempi nella sintassi CommonJS utilizzando le seguenti linee guida:

- Rimuovi `"type" : "module"` dall'ambiente `package.json` del tuo progetto.
- Converti tutte le ES6 `import` istruzioni in 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 ES6 `export` istruzioni in 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 CommonJS che in ES6 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();
```

AWS Elemental MediaConvert esempi

AWS Elemental MediaConvert è un servizio di transcodifica video basato su file con funzionalità di livello broadcast. 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 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

- Creazione e gestione di lavori di transcodifica in MediaConvert
- Utilizzo dei modelli di lavoro in MediaConvert

Creazione e gestione di lavori di transcodifica in MediaConvert

Questo esempio di codice di Node.js illustra:

- 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 AWS SDKs and Tools Reference Guide.
- 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 MediaConvert utente.
- 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 MediaConvert utente.

> **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 ES6Sintassi /CommonJS

Definizione di un semplice processo di transcodifica

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 MediaConvert console 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 lavori, *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/ «, *INPUT_BUCKET_AND_FILENAME* e 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, immettete 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 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, immettere 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 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 AWS Elemental MediaConvert di lavoro.
- 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 AWS SDKs and Tools Reference Guide.
- 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 MediaConvert utente.

> **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 ES6Sintassi /CommonJS

Creazione di un modello di lavoro

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

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 per verificare, *`KEY_PAIR_NAME`* con, con, *`TEMPLATE_NAME`* con *`ROLE_ARN`* 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, inserisci 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 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 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.

AWS Lambda esempi

AWS Lambda è un servizio di elaborazione serverless che consente di eseguire codice senza effettuare il provisioning o la gestione di server, creare una logica di scalabilità del cluster che riconosca il carico di lavoro, mantenere integrazioni di eventi o gestire i runtime.

L'API for è esposta tramite la classe client. JavaScript AWS Lambda LambdaService

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' JavaScript API 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 AWS SDKs and Tools Reference Guide.
- Crea un ruolo utente Amazon Cognito AWS Identity and Access Management (IAM) non autenticato pollySynthesizeSpeech : permessi e un pool di identità Amazon Cognito a cui è associato il ruolo IAM. La Crea le risorse utilizzando il AWSCloudFormation 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
>
> JSON

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

```
      "Effect": "Allow"
    },
    {
      "Action": "polly:SynthesizeSpeech",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Crea le risorse utilizzando il AWSCloudFormation

CloudFormation consente di creare e fornire implementazioni di AWS infrastrutture in modo prevedibile e ripetuto. Per ulteriori informazioni in merito CloudFormation, consulta la Guida per l'utente.AWS CloudFormation

Per creare lo 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 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.

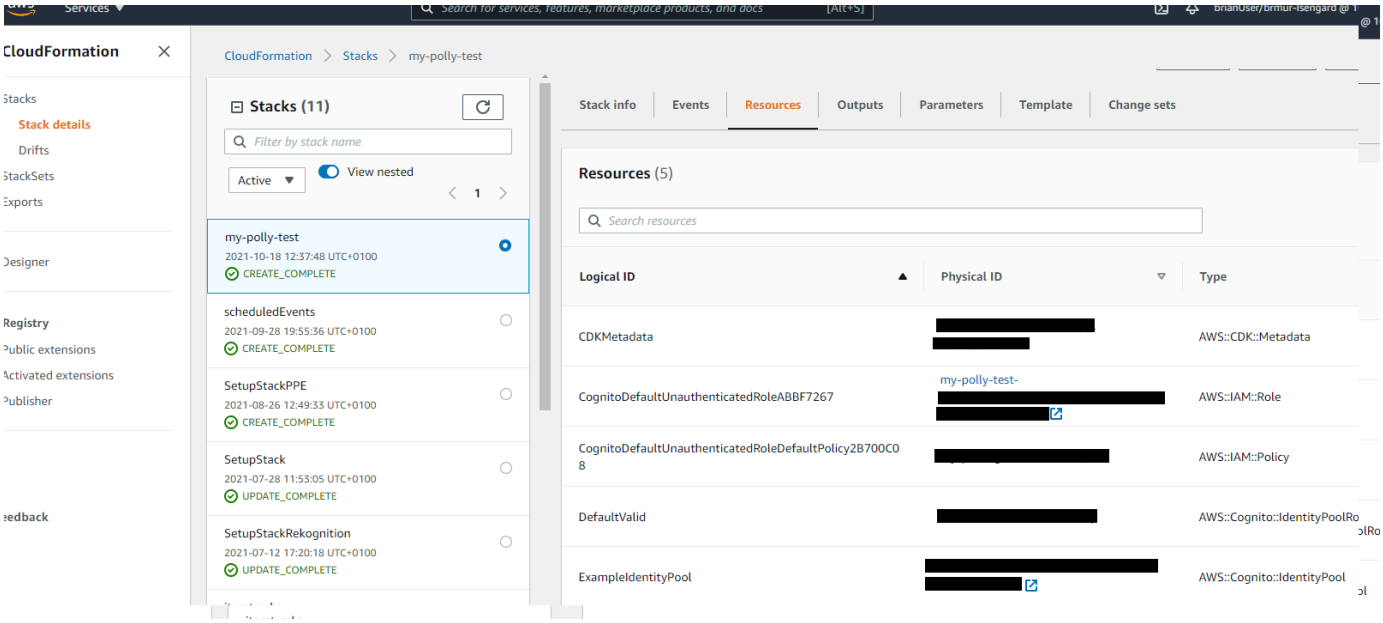
3. Esegui il comando seguente dalla riga di comando, sostituendolo *`STACK_NAME`* con un nome univoco per lo stack.

   > **Important**
   >
   > Il nome dello stack deve essere univoco all'interno di una AWS regione e AWS di un account. 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'CloudFormation utente.

4. Accedi alla console di 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 il*`REGION`*, e il*`BUCKET_NAME`*. Per accedere ad Amazon Polly, crea un oggetto di servizio `Polly` client. Sostituisci *`"IDENTITY_POOL_ID"`* con la `IdentityPoolId` pagina Sample del pool di identità 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.

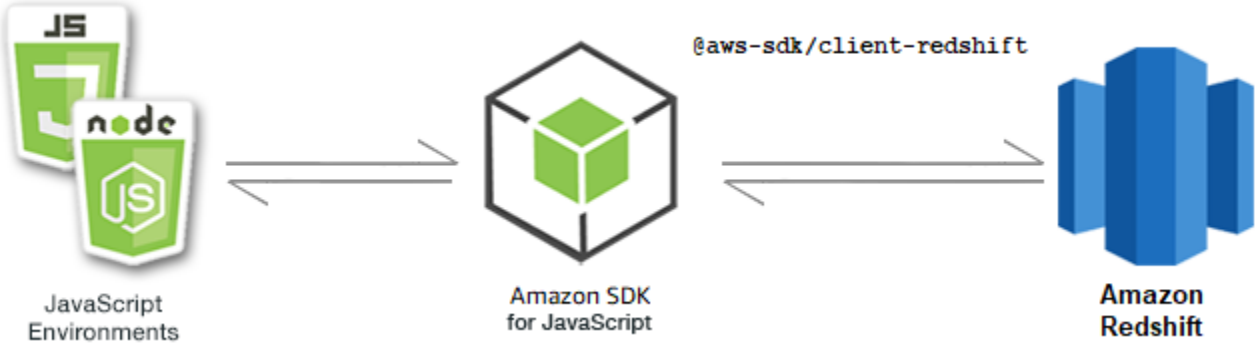
```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "./libs/pollyClient.js";
```

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

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

Questo codice di esempio può essere trovato qui su. GitHub

Esempi di Amazon Redshift

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



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

Argomenti

Esempi di Amazon Redshift

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 AWS SDKs and Tools Reference Guide.

> **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 ES6Sintassi /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 stai 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. *`REGION`*Sostituiscilo 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 su 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`* specificare il tipo di nodo da fornire, ad esempio 'dc2.large', ad esempio. *`MASTER_USERNAME`*e *`MASTER_USER_PASSWORD`* sono le credenziali di accesso dell'utente principale dell'istanza DB nel cluster. Per*`CLUSTER_TYPE`*, inserisci il tipo di cluster. Se lo specificate`single-node`, non è necessario il `NumberOfNodes` parametro. 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. *`REGION`*Sostituiscilo 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 su 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**
>
> Sostituisci *`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, immettere 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. *`REGION`*Sostituiscilo 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 su 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, immettere 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. *REGION*Sostituiscilo 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 su 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 specifichi 'false', specifica l'id dell'istantanea finale del cluster in. *`CLUSTER_SNAPSHOT_ID`* Puoi 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 Istantanee. Nota che lo stem non `rs:` fa parte dell'ID dell'istantanea.

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

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

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

```
    console.log("Error", err);
  }
};
run();
```

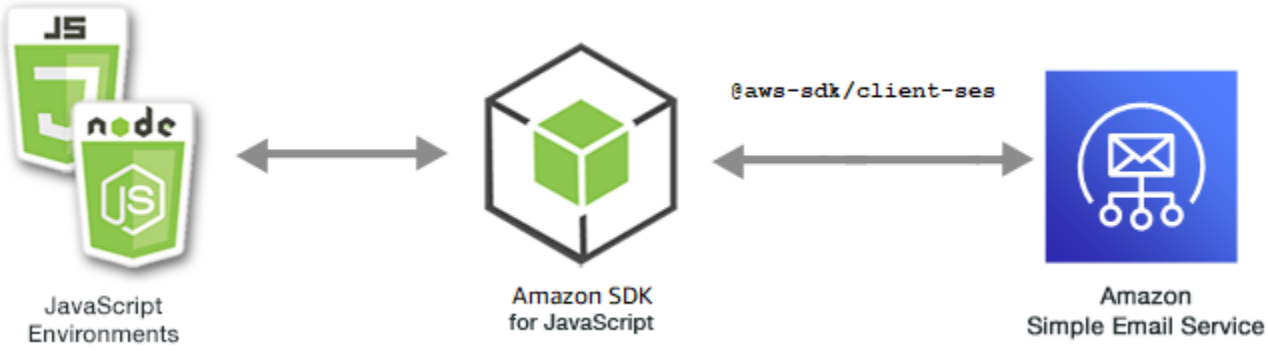
Per eseguire l'esempio, immettete quanto segue al prompt dei comandi.

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

Questo codice di esempio può essere trovato qui. GitHub

Esempi di Amazon Simple Email Service

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



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

Argomenti

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

Gestione delle identità Amazon SES

Questo esempio di codice di Node.js illustra:

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

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

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

Lo scenario

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

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

Attività prerequisite

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

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

> **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 ES6Sintassi /CommonJS

Elencare le tue identità

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

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

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

> **Note**
>
> Sostituisci *`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 Amazon SES.

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

> **Note**
>
> Sostituiscilo *`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 Amazon SES.

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

Note

Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

Note

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 Amazon SES.

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

Crea un oggetto per trasferire il parametro `Identity` per il metodo `DeleteIdentityCommand` della classe client `SES`. Per chiamare il `DeleteIdentityCommand` metodo, crea un oggetto del servizio client Amazon SES `request` per richiamare un oggetto del servizio client Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

> **Note**
>
> 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.

Utilizzo dei modelli di e-mail in Amazon SES

Questo esempio di codice di Node.js illustra:

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

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

Lo scenario

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

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

Attività prerequisite

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

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

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella AWS SDKs and Tools Reference Guide.

> **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 ES6Sintassi /CommonJS

Elencare i tuoi modelli di email

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

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare 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, utilizza un modulo Node.js per ottenere un modello di e-mail da utilizzare con Amazon SES.

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

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

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

const TEMPLATE_NAME = getUniqueName("TemplateName");

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

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

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (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, immettete quanto segue al prompt dei comandi. Amazon SES restituisce i dettagli del modello.

```
node ses_gettemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Creazione di un modello di email

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

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio

utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

> **Note**
>
> *TEMPLATE_NAME*Sostituitelo con un nome per il nuovo modello, *HtmlPart* con il contenuto del messaggio di posta elettronica con tag HTML 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, immettete quanto segue al prompt dei comandi. Il modello viene aggiunto ad Amazon SES.

```
node ses_createtemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Aggiornamento di un modello di e-mail

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

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

> **Note**
>
> Sostituiscilo *TEMPLATE_NAME* con il nome del modello e *HTML_PART* con il contenuto del messaggio di posta elettronica con tag HTML.

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

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

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

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

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

```
};
```

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

```
node ses_updatetemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Eliminazione di un modello di e-mail

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

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

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

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

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

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

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

```
node ses_deletetemplate.js
```

Questo codice di esempio può essere trovato qui su GitHub.

Invio di e-mail tramite Amazon SES

Questo esempio di codice di Node.js illustra:

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

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

Lo scenario

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

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

Attività prerequisite

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

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

- Creazione di un file di configurazione condiviso con le credenziali utente. Per ulteriori informazioni sulla fornitura di un file di credenziali condiviso, consulta File di configurazione e credenziali condivisi nella AWS SDKs and Tools Reference Guide.

> **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 ES6Sintassi /CommonJS

Requisiti per l'invio di messaggi e-mail

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

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

Invio di un'e-mail

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

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

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
```

```
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Questo codice di esempio può essere trovato qui GitHub.

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

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

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

> **Note**
>
> 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 viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendemail.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Invio di un'e-mail utilizzando un modello

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

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

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo in uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

> **Note**
>
> 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 viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendtemplatedemail.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Invio di e-mail in blocco utilizzando un modello

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

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

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

Questo codice di esempio può essere trovato qui GitHub.

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

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

il `SendBulkTemplatedEmailCommand` metodo, richiama un oggetto di servizio Amazon SES, passando i parametri.

> **Note**
>
> Questo esempio importa e utilizza i client del pacchetto AWS Service V3 richiesti, i comandi V3 e utilizza il `send` metodo secondo uno schema async/await. È possibile creare questo esempio utilizzando i comandi V2 invece apportando alcune modifiche minori. Per informazioni dettagliate, consultare Utilizzo dei comandi v3.

> **Note**
>
> 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 viene messa in coda per l'invio da parte di Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Questo codice di esempio può essere trovato qui su. GitHub

Esempi di servizi di notifica Amazon Simple

Amazon Simple Notification Service (Amazon SNS) è un servizio Web che coordina e gestisce la consegna o l'invio di messaggi a endpoint o client 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 SQS AWS Lambda , funzioni) utilizzano o ricevono il messaggio o la notifica tramite uno dei protocolli supportati (Amazon SQS, HTTP/S, e-mail AWS 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 AWS SDKs and Tools Reference Guide.

Important

Questi esempi mostrano come import/export client gli oggetti e i comandi del servizio 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 ES6Sintassi /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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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, immettere quanto segue al prompt dei comandi.

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

Questo codice di esempio è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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: '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,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
```

```
  //   }
  // }
  return response;
};
```

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

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

Questo codice di esempio è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 è disponibile 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 AWS SDKs and Tools Reference Guide.

> **Important**
>
> Questi esempi mostrano come import/export client gli oggetti e i comandi del servizio 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 ES6Sintassi /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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 Set SMSAttributes.

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, immettere quanto segue al prompt dei comandi.

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

Questo codice di esempio è disponibile 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 AWS SDKs and Tools Reference Guide.

> **Important**
>
> Questi esempi mostrano come import/export client gli oggetti e i comandi del servizio 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 ES6Sintassi /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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 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 è disponibile qui su GitHub.

Sottoscrizione di un indirizzo e-mail a un argomento

In questo esempio, usa un modulo Node.js per iscrivere 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo con la tua AWS regione.

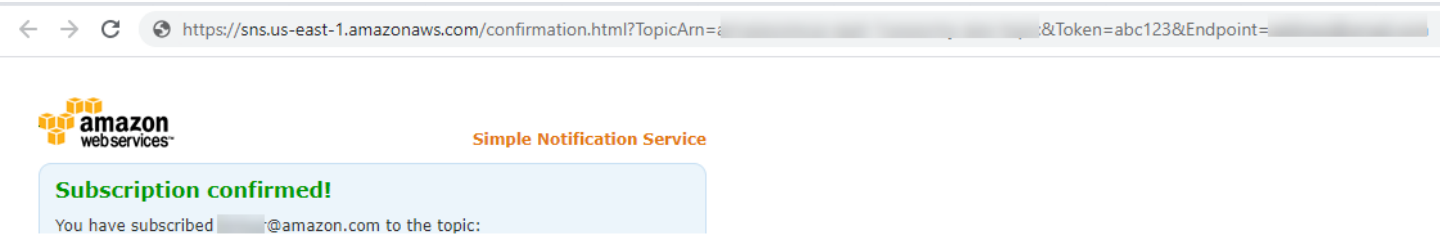
```
import { SNSClient } from "@aws-sdk/client-sns";

// 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 il valore 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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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**
>
> Sostituiscilo *`TOPIC_ARN`* con l'Amazon Resource Name (ARN) per l'argomento e *`MOBILE_ENDPOINT_ARN`* con l'endpoint a cui ti stai abbonando all'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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 è disponibile 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 AWS SDKs and Tools Reference Guide.

> **Important**
>
> Questi esempi mostrano come import/export client gli oggetti e i comandi del servizio 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 ES6Sintassi /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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 Set SMSAttributes in the Amazon Simple Notification Service API Reference.

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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 Set SMSAttributes in the Amazon Simple Notification Service API Reference.

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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 con il numero *PHONE_NUMBER* 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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 è disponibile 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 al suo interno, per creare l'oggetto client Amazon SNS. *`REGION`*Sostituiscilo 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 statunitense in formato E.164 apparirebbe come XXX555 +1001 0100.

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 è disponibile 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 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 ES6Sintassi /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.

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. Sostituiscilo *`REGION`* con la tua regione. AWS

```
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 su 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`* specificare il bucket Amazon S3 in cui salvare l'output. Per *`JOB_TYPE`* specificare tipi di lavoro. Per *`SOURCE_LOCATION`* specificare la posizione del

file sorgente. Per *SOURCE_FILE_LOCATION* specificare 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. Sostituiscilo *`REGION`* con la tua regione. AWS

```
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 su 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 lavoro 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, immettere 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. Sostituiscilo *`REGION`* con la tua regione. AWS

```
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 su 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**
>
> Sostituisci *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, immettere 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 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 ES6Sintassi /CommonJS

Avvio di un lavoro di trascrizione medica con Amazon Transcribe

Questo esempio dimostra come avviare un processo di trascrizione medica di Amazon Transcribe utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni, vedere startMedicalTranscriptionJob.

Crea una `libs` directory e crea un modulo Node.js con il nome del file`transcribeClient.js`. Copia e incolla il codice seguente al suo interno, per creare l'oggetto client Amazon Transcribe. Sostituiscilo *`REGION`* con la tua regione. AWS

```
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 su 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**
>
> Sostituiscilo *`MEDICAL_JOB_NAME`* con un nome per il lavoro di trascrizione medica. Per *`OUTPUT_BUCKET_NAME`* specificare il bucket Amazon S3 in cui salvare l'output. Per *`JOB_TYPE`* specificare tipi di lavoro. Per *`SOURCE_LOCATION`* specificare la posizione del file sorgente. Per *`SOURCE_FILE_LOCATION`* specificare la posizione del file multimediale di input.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
```

```
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

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

```
node transcribe-create-medical-job.js
```

Questo codice di esempio può essere trovato qui. GitHub

Pubblicazione di offerte di lavoro nel settore medico in Amazon Transcribe

Questo esempio mostra come elencare i lavori di trascrizione di Amazon Transcribe utilizzando il. AWS SDK for JavaScript Per ulteriori informazioni, consulta ListTranscriptionMedicalJobsCommand.

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. Sostituiscilo *`REGION`* con la tua regione. AWS

```
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 su 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. Sostituiscilo *`REGION`* con la tua regione. AWS

```
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 su 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, immettere 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' EC2 istanza Amazon

Uno scenario comune per l'utilizzo di Node.js con l'SDK per JavaScript consiste nel configurare ed eseguire un'applicazione Web Node.js su un'istanza Amazon Elastic Compute Cloud EC2 (Amazon). In questo tutorial, creerai un'istanza Linux, ti connetterai a essa tramite SSH, quindi installerai Node.js per l'esecuzione su tale istanza.

Prerequisiti

Questo tutorial presuppone che tu abbia già avviato un'istanza Linux con un nome DNS pubblico raggiungibile da Internet e alla quale puoi connetterti tramite 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. Connettersi all'istanza Linux come `ec2-user` tramite SSH.
2. Installa node version manager (`nvm`) digitando quanto segue nella riga di comando.

   > **Warning**
   >
   > AWS non controlla il codice seguente. Prima di eseguirlo, assicurati di verificarne l'autenticità e l'integrità. Ulteriori informazioni su questo codice sono disponibili nel repository nvm. GitHub

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
   ```

   Useremo `nvm` per installare Node.js perché `nvm` può installare più versioni di Node.js e permetterti di passare da una all'altra.
3. Carica `nvm` digitando quanto segue nella riga di comando.

   ```
   source ~/.bashrc
   ```
4. Usa nvm per installare l'ultima versione LTS di Node.js digitando quanto segue nella riga di comando.

   ```
   nvm install --lts
   ```

   L'installazione di Node.js installa anche il Node Package Manager (`npm`) in modo da poter installare moduli aggiuntivi secondo necessità.
5. Verificare che Node.js sia installato e correttamente in esecuzione digitando quanto segue nella riga di comando.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Viene visualizzato il seguente messaggio che mostra la versione di Node.js in esecuzione.

   ```
   Running Node.js VERSION
   ```

> **Note**
>
> L'installazione del nodo si applica solo alla EC2 sessione Amazon corrente. Se si riavvia la sessione CLI, è necessario utilizzare nuovamente nvm per abilitare la versione del nodo installata. Se l'istanza viene terminata, devi installare nuovamente il nodo. L'alternativa è creare un'Amazon Machine Image (AMI) dell' EC2 istanza Amazon una volta ottenuta la configurazione che desideri mantenere, come descritto nel seguente argomento.

Creazione di un'Amazon Machine Image (AMI)

Dopo aver installato Node.js su un' EC2 istanza Amazon, puoi creare un'Amazon Machine Image (AMI) da quell'istanza. La creazione di un'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 Creazione di un'AMI Linux supportata da Amazon EBS 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.

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 APIs HTTP e su larga scala. Gli sviluppatori di API possono creare APIs quell'accesso AWS o altri servizi Web, oltre ai dati archiviati nel cloud. AWS In qualità di sviluppatore di API Gateway, puoi creare applicazioni client APIs per utilizzarle nelle tue applicazioni client. Per ulteriori informazioni, consulta Cos'è Amazon API Gateway.

AWS Lambda è 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 What is AWS Lambda.

In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Ad esempio, si supponga 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.

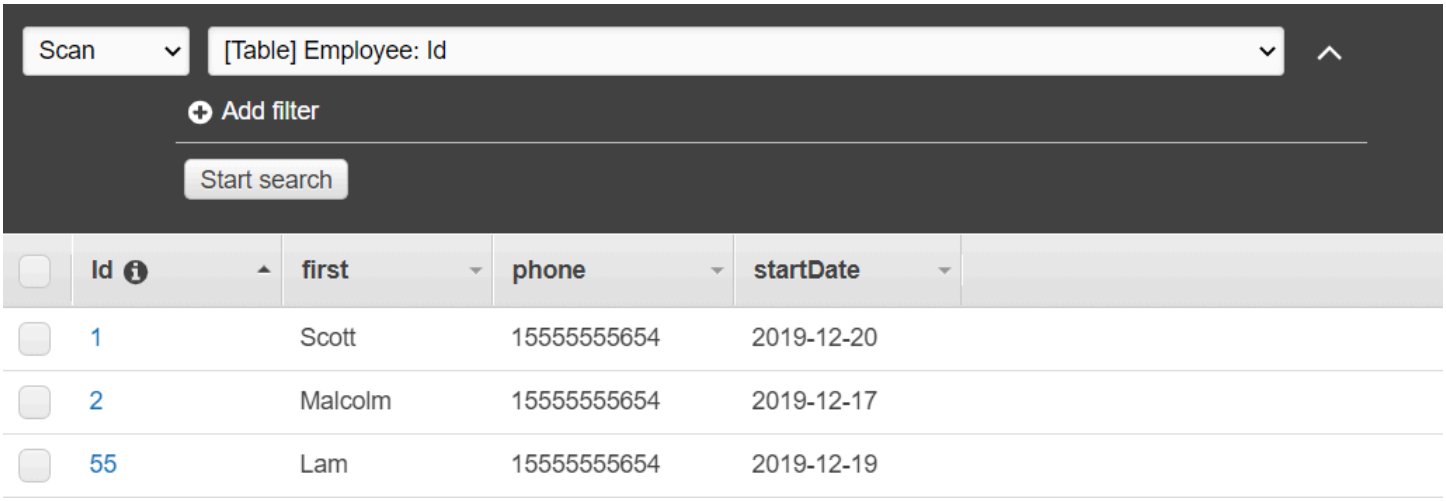
Il completamento dell'esempio dovrebbe richiedere circa 20 minuti.

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 AWS SDKs and Tools Reference Guide.

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 utilizzando il metodo CloudFormation descritto in questo tutorial.

Crea le AWS risorse utilizzando CloudFormation

CloudFormation consente di creare e fornire implementazioni di AWS infrastrutture in modo prevedibile e ripetuto. Per ulteriori informazioni in merito CloudFormation, consulta la Guida per l'utente.AWS CloudFormation

Per creare lo CloudFormation stack utilizzando: AWS CLI

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 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, sostituendolo *`STACK_NAME`* con un nome univoco per lo stack.

> **Important**
>
> Il nome dello stack deve essere univoco all'interno di una AWS regione e AWS di un account. 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'CloudFormation utente.

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 AWS Lambda funzione

Configurazione dell'SDK

Nella `libs` directory, crea file denominati `snsClient.js` 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 la AWS regione. 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 regione. 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` Sostituitelo *`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.)

```
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 async/await funzione 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.)

```
// 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));
    for (const element of data.Items) {
      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 la 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. Sostituire *`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

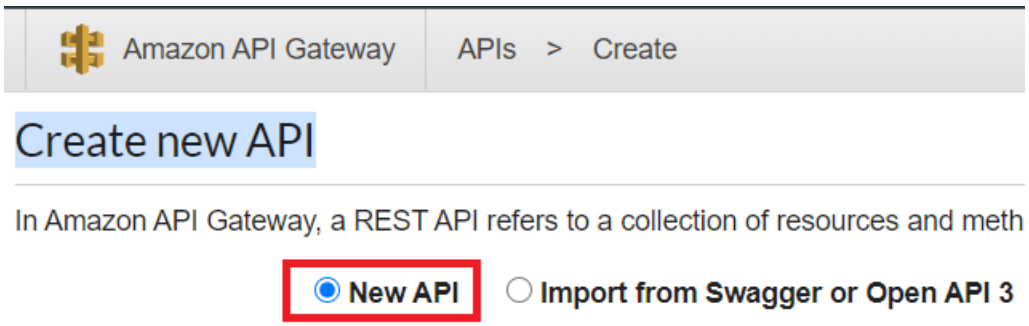
Per creare l'app:

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

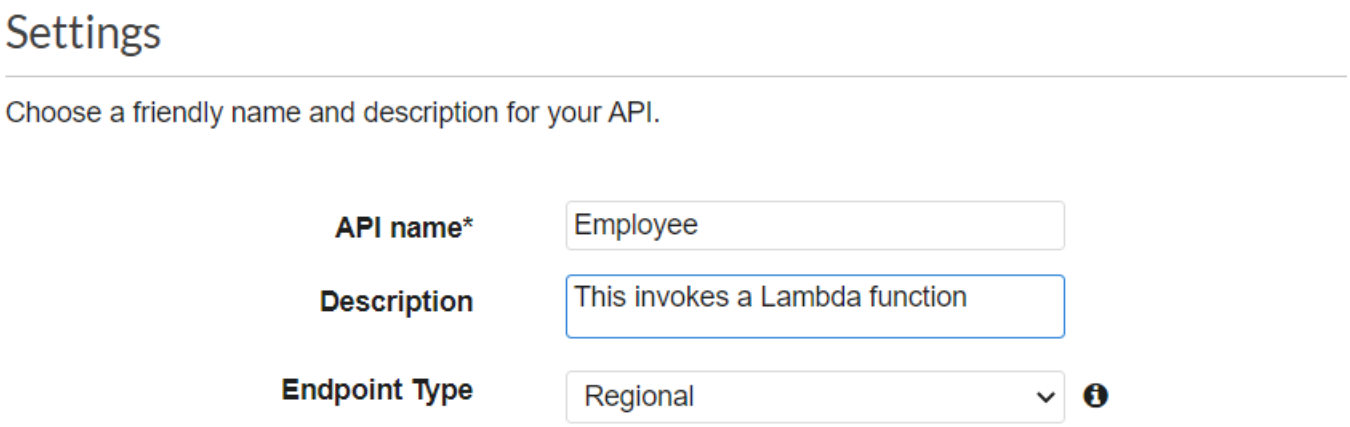
Crea la restante API

Puoi utilizzare la console API Gateway per creare un endpoint REST per la funzione Lambda. Una volta terminata, puoi richiamare la funzione Lambda utilizzando una chiamata restful.

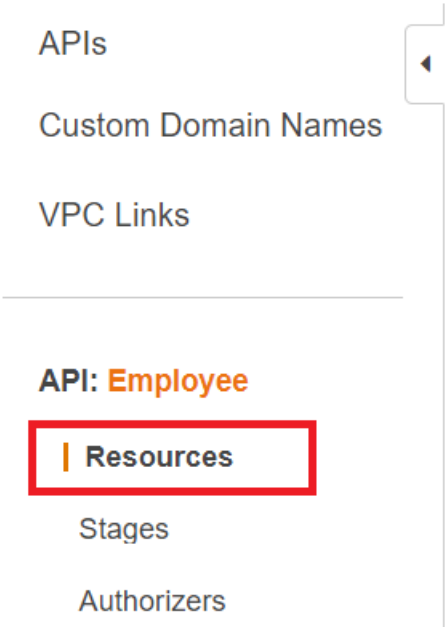
1. Accedi alla console Amazon API Gateway.
2. In Rest API, scegli Build.
3. Seleziona Nuova API.



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

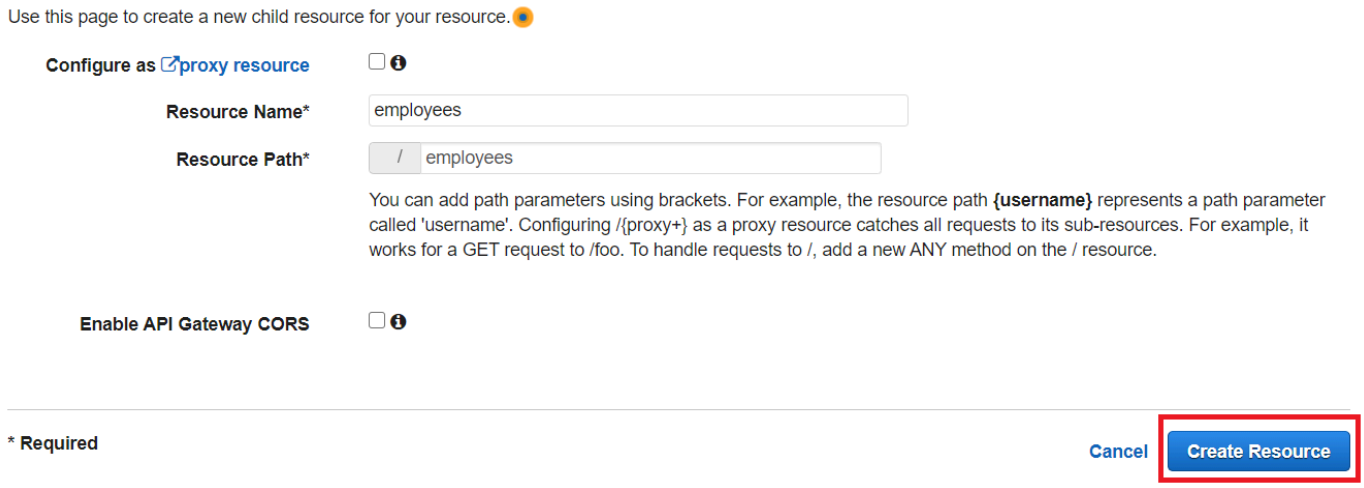


5. Seleziona Create API (Crea API).
6. Scegli Risorse nella sezione Dipendenti.

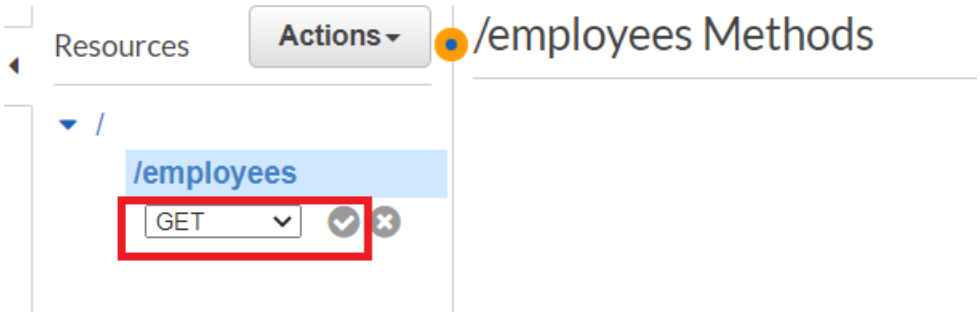


7. Nel campo del nome, specifica i dipendenti.

8. Selezionare Create Resources (Crea risorse).
9. Dal menu a discesa Azioni, scegli Crea risorse.



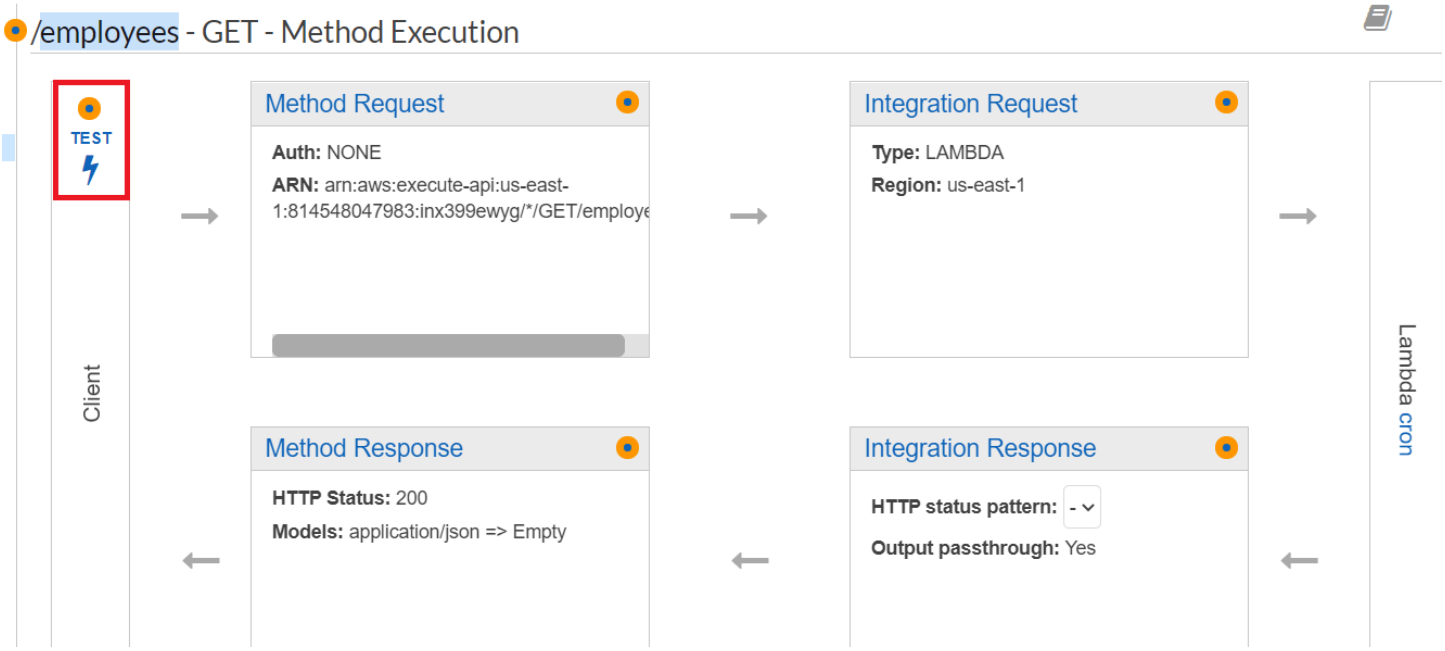
10. Scegli /employees, seleziona Crea metodo dal menu Azioni, quindi seleziona GET dal menu a discesa sotto /employees. Selezionare l'icona del segno di spunta.



11. Scegliete la funzione Lambda e immettete mylambdafunction come nome della funzione Lambda. Scegli Save (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.

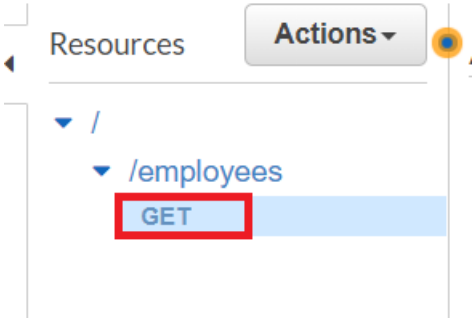


Una volta richiamata la funzione Lambda, è possibile visualizzare il file di registro per visualizzare un messaggio di successo.

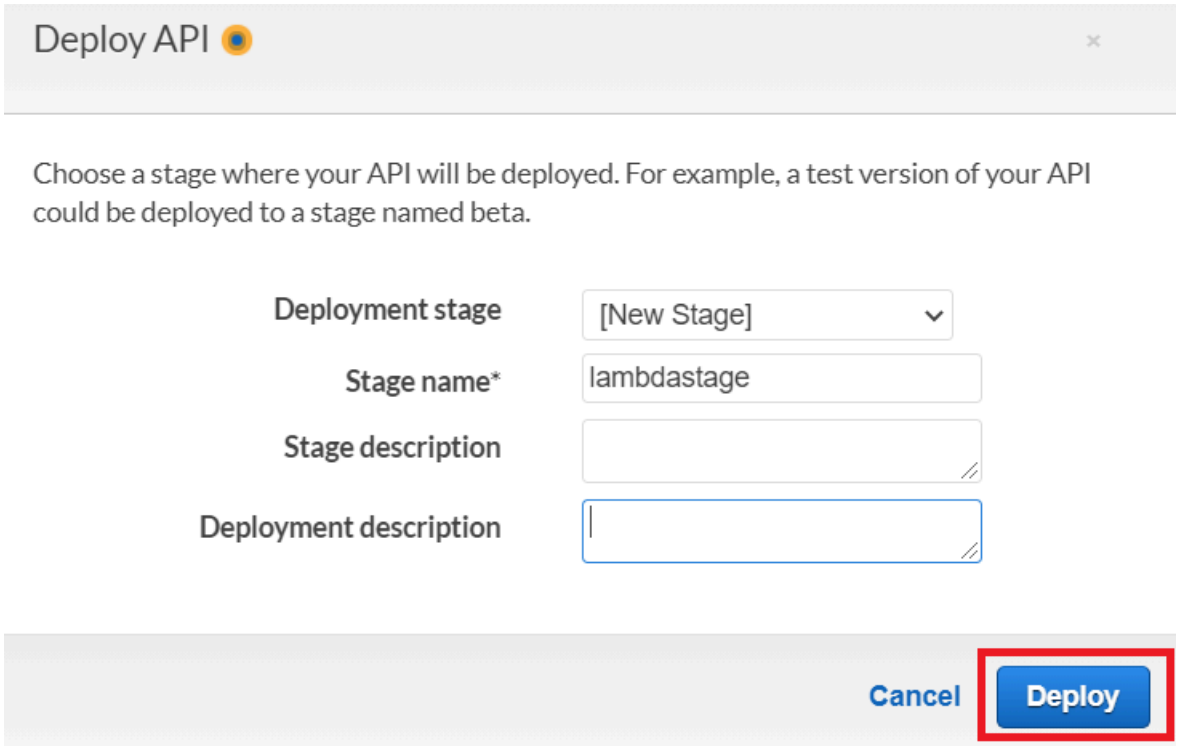
Implementa il metodo API Gateway

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

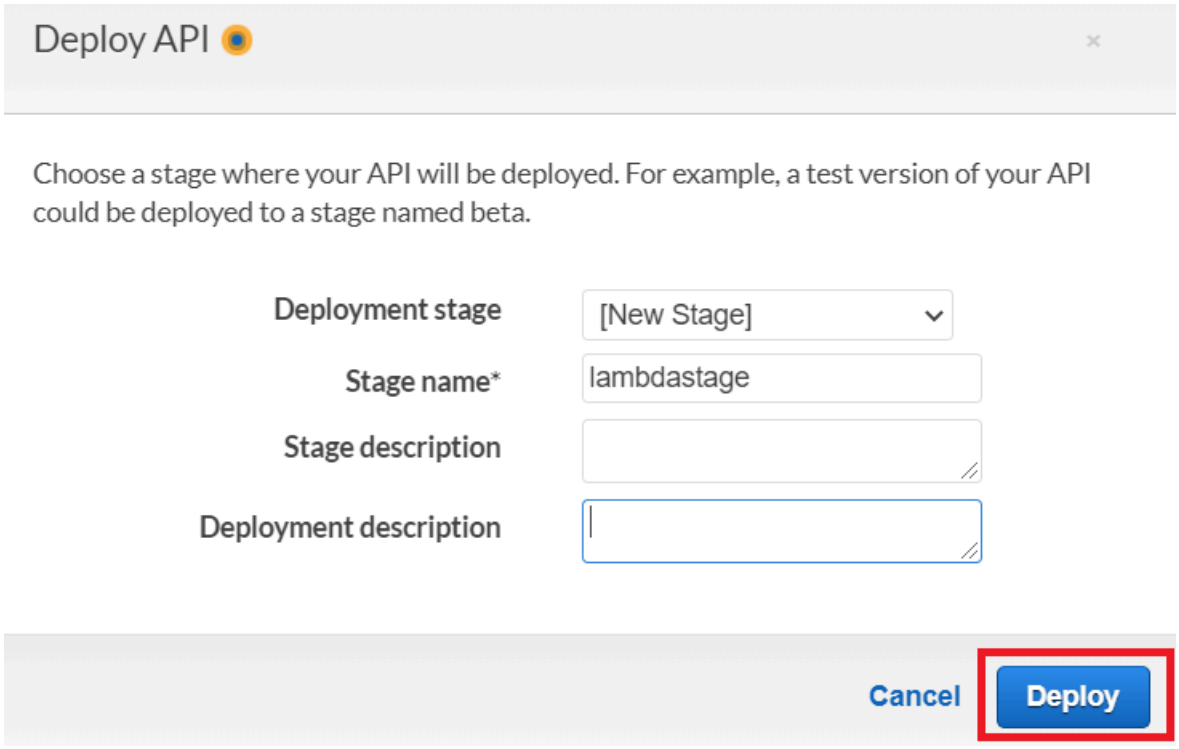
1. Scegli Get.



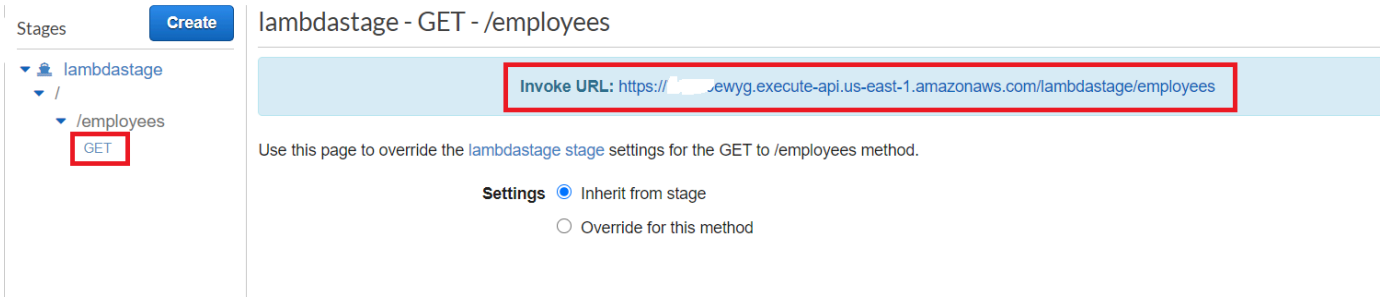
2. Dal menu a discesa Azioni, seleziona Deploy API.



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 CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

1. Apri il file CloudFormation nella 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 Lambda è 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 What is AWS Lambda.

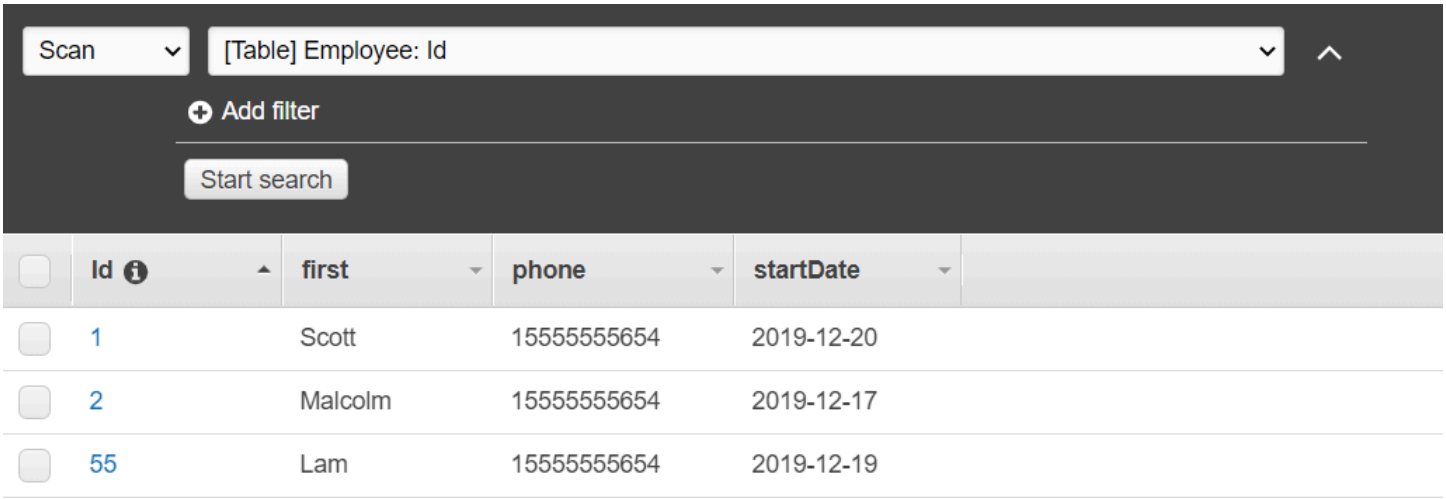
In questo tutorial, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Ad esempio, si supponga che un'organizzazione invii un messaggio di testo mobile ai propri dipendenti per congratularsi con loro 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 AWS SDKs and Tools Reference Guide.

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 usando CloudFormation come descritto in questo tutorial.

Crea le AWS risorse utilizzando CloudFormation

CloudFormation consente di creare e fornire implementazioni di AWS infrastrutture in modo prevedibile e ripetuto. Per ulteriori informazioni in merito CloudFormation, consulta la Guida per l'utente.AWS CloudFormation

Per creare lo CloudFormation stack utilizzando: AWS CLI

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 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, sostituendolo *`STACK_NAME`* con un nome univoco per lo stack.

   > **Important**
   >
   > Il nome dello stack deve essere univoco all'interno di una AWS regione e AWS di un account. 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'CloudFormation utente.

Visualizza un elenco delle risorse nella console aprendo lo stack sulla 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 popolare la tabella, crea prima una directory denominata`libs`, in essa crea un file denominato `dynamoClient.js` e incolla 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 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 AWS Lambda funzione

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 con la regione*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 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";
// 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 async/await funzione 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 la versione ZIP della tua funzione Lambda. Sostituisci *IAM_ROLE_ARN* con l'Amazon Resource Number (ARN) del ruolo IAM che hai creato nell'Crea le risorse AWSargomento di questo tutorial. Sostituire *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. Scegliere Aggiungi.

> **Note**
>
> Per ulteriori informazioni, consulta Uso di Lambda con CloudWatch gli eventi.

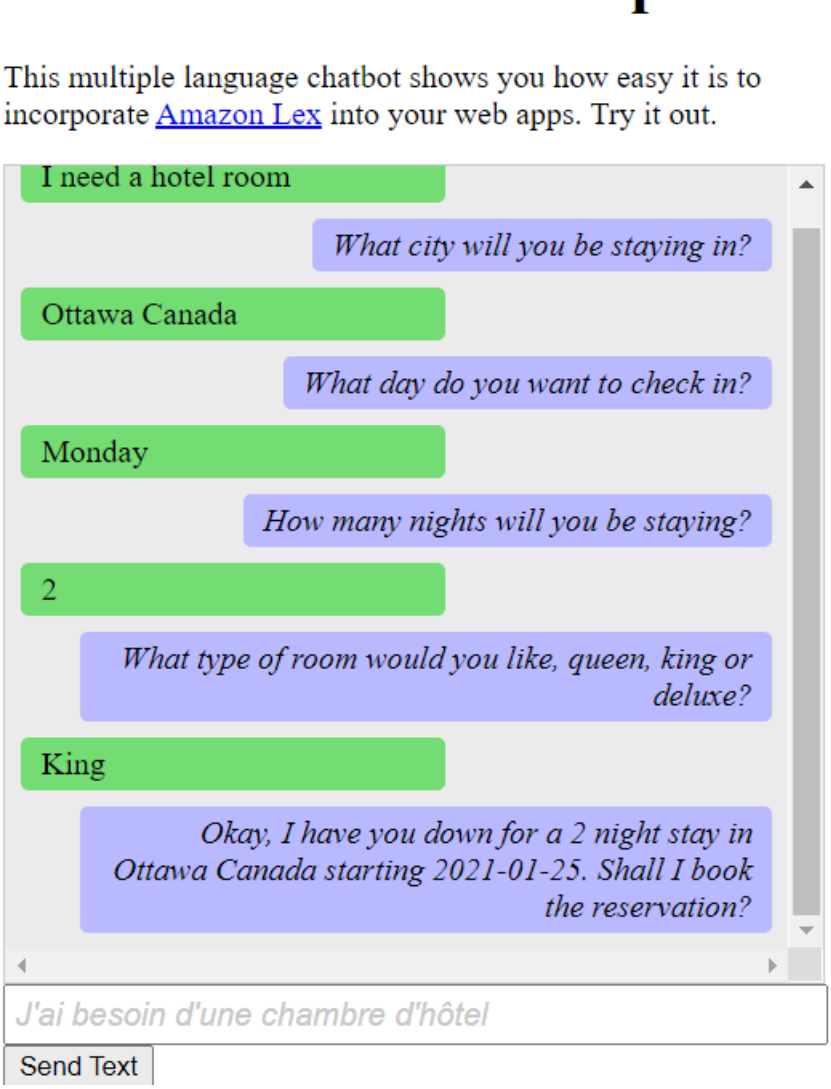
Eliminare le risorse

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 CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

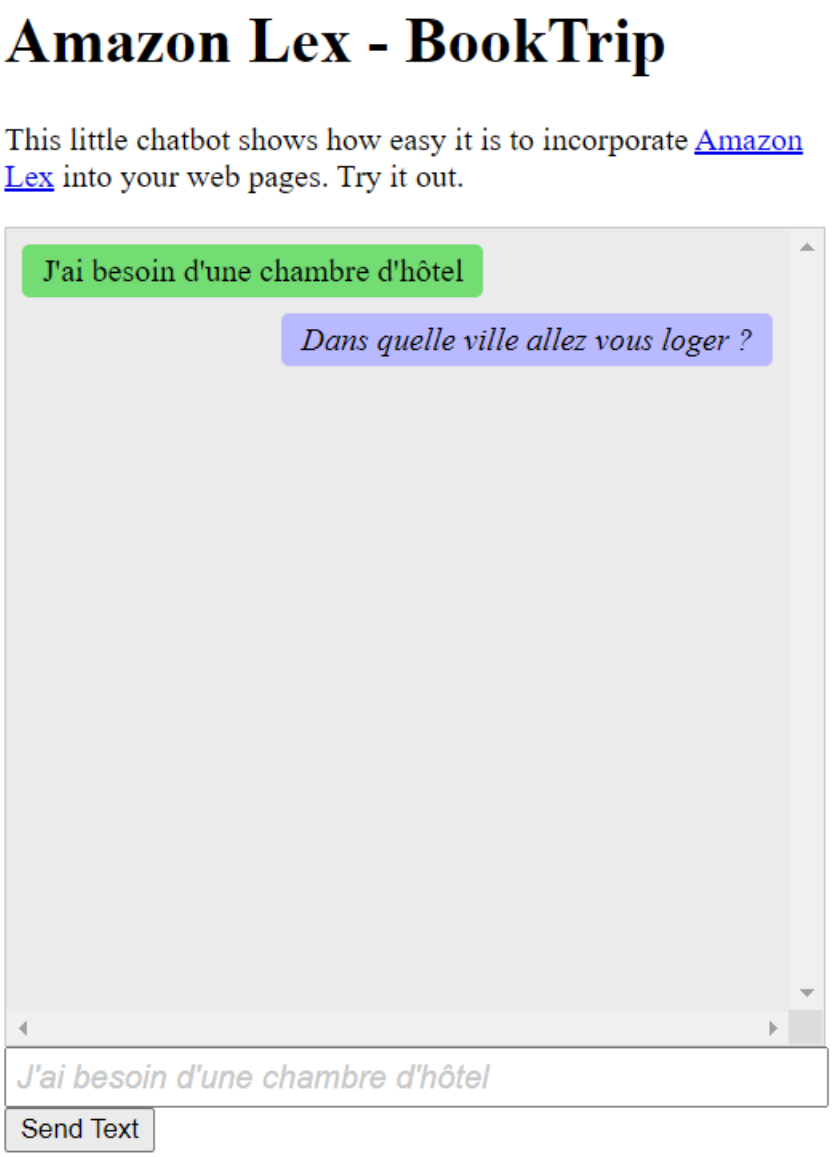
1. Apri la CloudFormation console.
2. Nella pagina Stacks, seleziona lo stack.
3. Scegli Delete (Elimina).

Creazione di un chatbot Amazon Lex

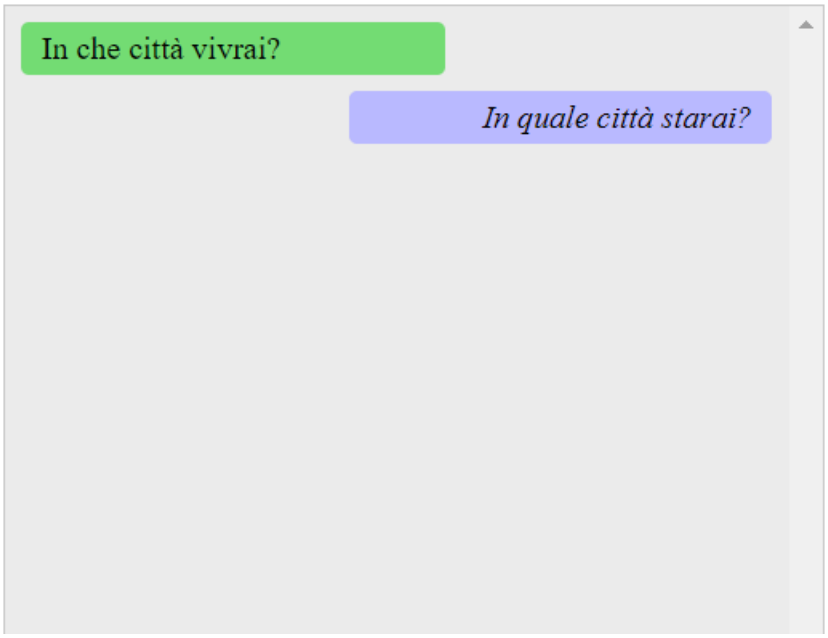
Puoi creare un chatbot Amazon Lex all'interno di un'applicazione Web per coinvolgere i visitatori del tuo sito Web. Un chatbot di Amazon Lex è una funzionalità che esegue conversazioni in chat online con gli utenti senza fornire un contatto diretto con una persona. Ad esempio, l'illustrazione seguente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

> **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 ES6Sintassi /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 CloudFormation

CloudFormation consente di creare e fornire implementazioni di AWS infrastrutture in modo prevedibile e ripetuto. Per ulteriori informazioni in merito CloudFormation, consulta la Guida per l'utente.AWS CloudFormation

Per creare lo CloudFormation stack utilizzando: AWS CLI

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 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, sostituendolo *`STACK_NAME`* con un nome univoco per lo stack.

   > **Important**
   >
   > Il nome dello stack deve essere univoco all'interno di una AWS regione e AWS di un account. 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'CloudFormation utente.

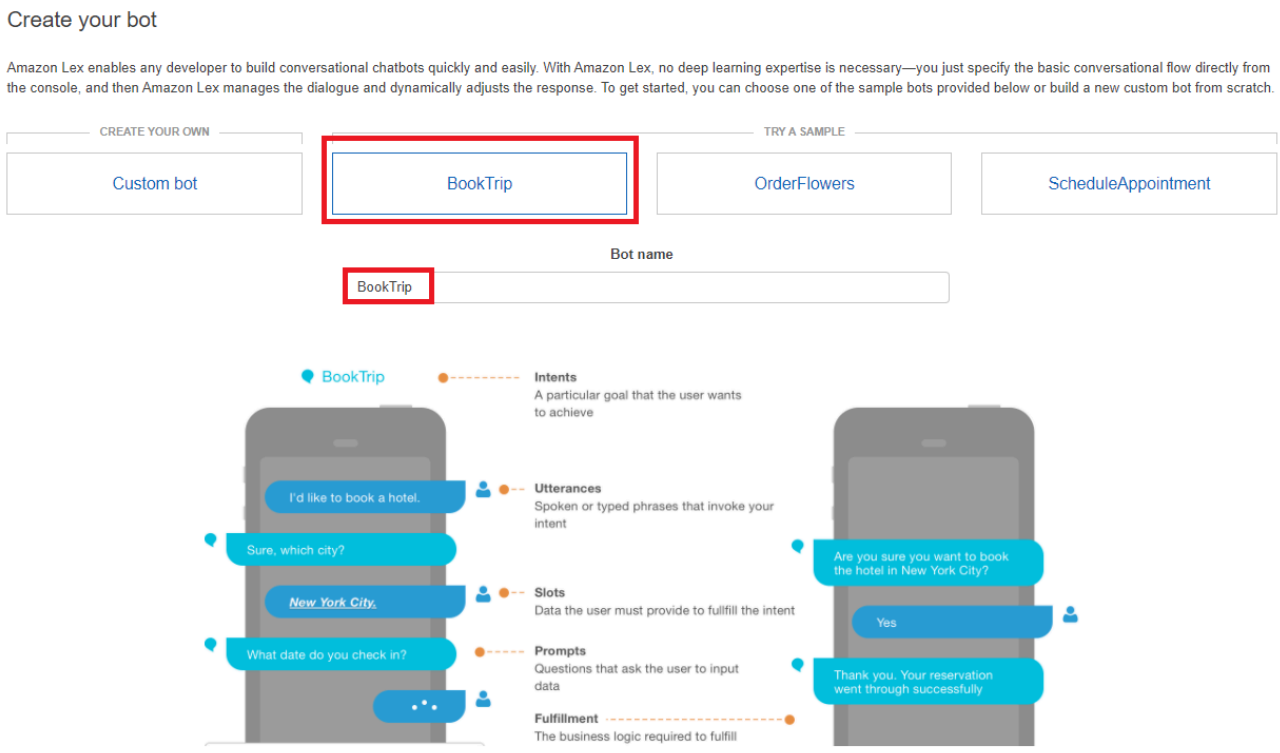
   Per visualizzare le risorse create, apri la console Amazon Lex, scegli lo stack e seleziona la scheda Risorse.

Creazione di un bot Amazon Lex

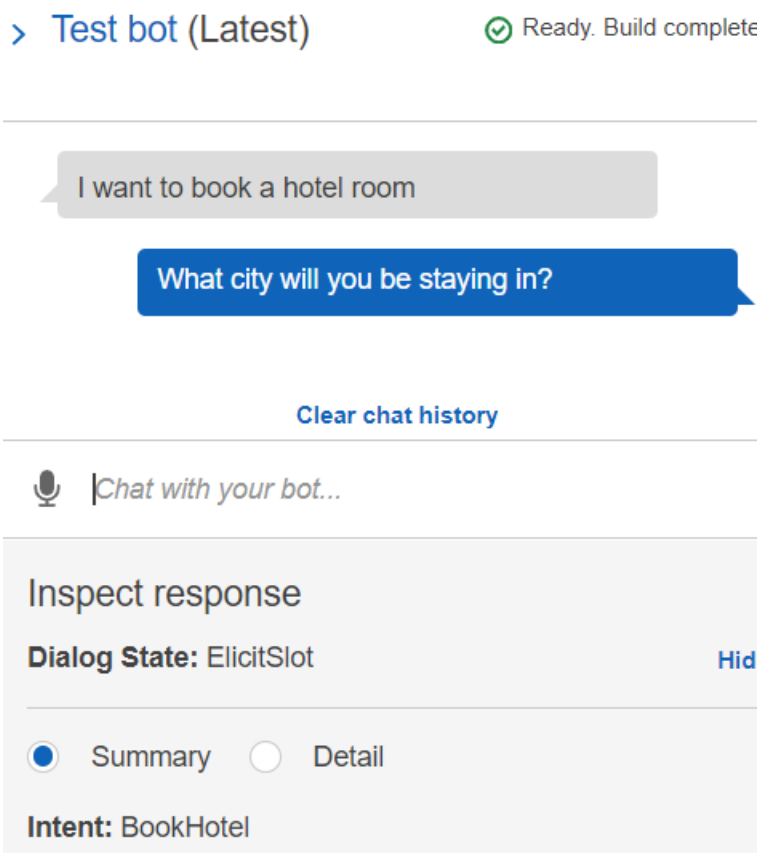
> **Important**
>
> Usa la versione 1 della console Amazon Lex per creare il bot. Questo esempio non funziona con i bot creati utilizzando V2.

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 di 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 il AWS SDK for JavaScript).

> **Note**
>
> Devi fare riferimento al nome del bot e all'alias del bot nel codice JavaScript .

Crea il codice HTML

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. Creerai questo file inCrea il codice 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 *`REGION`* con AWS 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 riportato di seguito in ciascuno di essi *`REGION`* e sostituiscilo con *`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 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 rispettivamente *`BOT_NAME`* con l'alias e il nome del tuo bot Amazon Lex e *`USER_ID`* con un ID utente. La funzione `createResponse` asincrona esegue le seguenti operazioni:

- 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";
```

```
let g_text = "";
// Set the focus to the input box.
document.getElementById("wisdom").focus();

function showRequest() {
  const conversationDiv = document.getElementById("conversation");
  const requestPara = document.createElement("P");
  requestPara.className = "userRequest";
  requestPara.appendChild(document.createTextNode(g_text));
  conversationDiv.appendChild(requestPara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function showResponse(lexResponse) {
  const conversationDiv = document.getElementById("conversation");
  const responsePara = document.createElement("P");
  responsePara.className = "lexResponse";

  const 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;
  const 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.
  const wisdomText = document.getElementById("wisdom");
  wisdomText.value = "";
  wisdomText.locked = false;
```

```
}

// Respond to user's input.
const createResponse = async () => {
  // Confirm there is text to submit.
  const wisdomText = document.getElementById("wisdom");
  if (wisdomText?.value && wisdomText.value.trim().length > 0) {
    // Disable input to show it is being sent.
    const 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);
          const 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
   ```

Fasi successive

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 CloudFormation stack che hai creato nell'Crea le risorse AWSargomento di questo tutorial, come segue:

1. Apri la 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.

SDK per esempi di JavaScript codice (v3)

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

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Alcuni servizi contengono ulteriori categorie di esempi che mostrano come sfruttare le librerie o le funzioni specifiche del servizio.

Servizi

- Esempi di API Gateway che utilizzano SDK for JavaScript (v3)
- Esempi di Aurora con 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 Amazon Bedrock Agents che utilizzano SDK for JavaScript (v3)
- Esempi di Amazon Bedrock Agents 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 identità di Amazon Cognito con SDK for JavaScript (v3)
- Esempi di Amazon Cognito Identity Provider con 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 con SDK JavaScript for (v3)
- EC2 Esempi di Amazon che utilizzano SDK per JavaScript (v3)
- Elastic Load Balancing - Esempi della versione 2 che utilizzano SDK for JavaScript (v3)
- AWS Entity Resolution esempi che utilizzano SDK for JavaScript (v3)
- EventBridge esempi che utilizzano SDK for JavaScript (v3)
- Esempi di Amazon Glacier con SDK JavaScript for (v3)
- AWS Glue esempi che utilizzano SDK for JavaScript (v3)
- HealthImaging esempi che utilizzano SDK for JavaScript (v3)
- Esempi IAM che utilizzano SDK for JavaScript (v3)
- AWS IoT SiteWise esempi che utilizzano SDK for JavaScript (v3)
- Esempi di Kinesis con SDK for JavaScript (v3)
- Esempi di Lambda con SDK for JavaScript (v3)
- Esempi di Amazon Lex con SDK for JavaScript (v3)
- Esempi di Amazon Location con SDK for JavaScript (v3)
- Esempi di Amazon MSK con SDK for JavaScript (v3)
- Esempi di Amazon Personalize utilizzando SDK for JavaScript (v3)
- Esempi di Amazon Personalize Events utilizzando SDK for JavaScript (v3)
- Esempi di Amazon Personalize Runtime utilizzando SDK for JavaScript (v3)
- Esempi di Amazon Pinpoint con SDK for JavaScript (v3)
- Esempi di Amazon Polly con SDK for JavaScript (v3)
- Esempi di Amazon RDS con 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 con SDK for (v3) JavaScript
- Esempi di Amazon S3 con SDK for JavaScript (v3)
- SageMaker Esempi di intelligenza artificiale che utilizzano SDK for (v3 JavaScript)
- Esempi di Secrets Manager con SDK for JavaScript (v3)
- Esempi di Amazon SES con SDK for JavaScript (v3)
- Esempi di Amazon SNS con SDK for JavaScript (v3)
- Esempi di Amazon SQS con SDK for JavaScript (v3)

- Esempi di Step Functions con SDK for JavaScript (v3)
- AWS STS esempi che utilizzano SDK for JavaScript (v3)
- Supporto esempi che utilizzano SDK for JavaScript (v3)
- Esempi di Systems Manager con SDK for JavaScript (v3)
- Esempi di Amazon Textract con SDK for JavaScript (v3)
- Esempi di Amazon Transcribe con SDK JavaScript for (v3)
- Esempi di Amazon Translate con SDK for JavaScript (v3)

Esempi di API 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) with API Gateway.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Scenari

Scenari

Creazione di un'applicazione serverless per gestire foto

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

SDK per JavaScript (v3)

> Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.
>
> Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

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

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Utilizzo di Gateway API per invocare una funzione Lambda

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

SDK per JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando l'API di JavaScript runtime Lambda. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda invocata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

Esempi di Aurora 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 Aurora.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

L'esempio di codice seguente mostra come creare un'applicazione web che traccia gli elementi di lavoro in database Amazon Aurora serverless e utilizza Amazon Simple Email Service (Amazon SES) per inviare report.

SDK per (v3 JavaScript)

> Mostra come utilizzare AWS SDK for JavaScript (v3) per creare un'applicazione Web che tenga traccia degli elementi di lavoro in un database Amazon Aurora e invii report tramite e-mail utilizzando Amazon Simple Email Service (Amazon SES). Questo esempio utilizza un front-end creato con React.js per interagire con un backend Express Node.js.
>
> - Integra un'applicazione web React.js con. Servizi AWS
> - Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
> - Invia un report per e-mail degli elementi di lavoro filtrati tramite Amazon SES.
> - Distribuisci e gestisci risorse di esempio con lo AWS CloudFormation script incluso.
>
> Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.
>
> Servizi utilizzati in questo esempio
>
> - Aurora

- Amazon RDS
- Servizi di dati di Amazon RDS
- Amazon SES

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 azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`AttachLoadBalancerTargetGroups`

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

SDK per JavaScript (v3)

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

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

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

- Per i dettagli sull'API, AttachLoadBalancerTargetGroupsconsulta AWS SDK for JavaScript API Reference.

Scenari

Creazione e gestione di un servizio resiliente

L'esempio di codice seguente 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 (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze in un intervallo specificato.
- Gestisci e distribuisci le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo Auto Scaling 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.

SDK per JavaScript (v3)

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

Esegui lo scenario interattivo al prompt dei comandi.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
```

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

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-
v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}
```

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),
      );
    }
    return MESSAGES.noIpRules;
  },
),
new ScenarioInput(
  "shouldAddInboundRule",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return false;
    }
    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);
    }
    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(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${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,
      );
    }
```

```
    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("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
      .replace("${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(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
    return MESSAGES.deletedPolicy.replace(
      "${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);
    }
    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,
      );
    }
    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,
      );
    }
    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,
      );
    }
    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(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${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(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${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(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${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);
    }
    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);
    }
    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");
    }
    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(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${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(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
    return MESSAGES.deletedSsmOnlyPolicy.replace(
      "${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,
      );
```

```
    }
    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,
      );
    }
    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;
    }
    console.log(err.name);
    throw err;
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
```

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer

- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Esempi di Amazon Bedrock con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Bedrock.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Azioni

Nozioni di base

Hello Amazon Bedrock

Il seguente esempio di codice mostra come iniziare a usare Amazon Bedrock.

SDK per JavaScript (v3)

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

```
import { fileURLToPath } from "node: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 (const model of models) {
    console.log("=".repeat(42));
    console.log(` Model: ${model.modelId}`);
    console.log("-".repeat(42));
    console.log(` Name: ${model.modelName}`);
    console.log(` Provider: ${model.providerName}`);
    console.log(` Model ARN: ${model.modelArn}`);
    console.log(` Input modalities: ${model.inputModalities}`);
    console.log(` Output modalities: ${model.outputModalities}`);
```

```
    console.log(` Supported customizations: ${model.customizationsSupported}`);
    console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
    console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
    console.log(`${"=".repeat(42)}\n`);
  }

  const active = models.filter(
    (m) => m.modelLifecycle.status === "ACTIVE",
  ).length;
  const legacy = models.filter(
    (m) => m.modelLifecycle.status === "LEGACY",
  ).length;

  console.log(
    `There are ${active} active and ${legacy} legacy foundation models in
 ${REGION}.`,
  );

  return response;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- Per i dettagli sull'API, ListFoundationModelsconsulta AWS SDK for JavaScript API Reference.

Azioni

`GetFoundationModel`

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

SDK per JavaScript (v3)

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

Ottiene i dettagli su un modello di fondazione.

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

import {
  BedrockClient,
  GetFoundationModelCommand,
} from "@aws-sdk/client-bedrock";

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 models.
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();

  const command = new GetFoundationModelCommand({
    modelIdentifier: "amazon.titan-embed-text-v1",
  });

  const response = await client.send(command);

  return response.modelDetails;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const model = await getFoundationModel();
  console.log(model);
}
```

- Per i dettagli sull'API, GetFoundationModelconsulta AWS SDK for JavaScript API Reference.

`ListFoundationModels`

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

SDK per JavaScript (v3)

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

Elenca i modelli di fondazione disponibili.

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

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

/**
 * List the available Amazon Bedrock foundation models.
 *
 * @return {FoundationModelSummary[]} - The list of available bedrock foundation
 models.
 */
export const listFoundationModels = async () => {
  const client = new BedrockClient();

  const input = {
    // byProvider: 'STRING_VALUE',
    // byCustomizationType: 'FINE_TUNING' || 'CONTINUED_PRE_TRAINING',
    // byOutputModality: 'TEXT' || 'IMAGE' || 'EMBEDDING',
    // byInferenceType: 'ON_DEMAND' || 'PROVISIONED',
  };

  const command = new ListFoundationModelsCommand(input);

  const response = await client.send(command);

  return response.modelSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
```

```
  const models = await listFoundationModels();
  console.log(models);
}
```

- Per i dettagli sull'API, ListFoundationModelsconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon Bedrock Runtime con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Bedrock Runtime.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Scenari
- Amazon Nova
- Amazon Nova Canvas
- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

Nozioni di base

Hello Amazon Bedrock

Il seguente esempio di codice mostra come iniziare a usare Amazon Bedrock.

SDK per JavaScript (v3)

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

```
/**
 * @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 "node: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 i dettagli sull'API, InvokeModelconsulta AWS SDK for JavaScript API Reference.

Scenari

Invocare più modelli di fondazione in 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

SDK per (v3) JavaScript

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

```
import { fileURLToPath } from "node: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}'...`),
);

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,
);

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 informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - InvokeModel
  - InvokeModelWithResponseStream

Utilizzo dello strumento con l'API Converse

Il seguente esempio di codice mostra come creare un'interazione tipica tra un'applicazione, un modello di intelligenza artificiale generativa e strumenti connessi o come APIs mediare le interazioni tra l'IA e il mondo esterno. Viene utilizzato l'esempio di collegamento di un'API esterna per il meteo al modello di IA in modo che vengano fornite informazioni meteorologiche in tempo reale in base sull'input dell'utente.

SDK per (v3 JavaScript)

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

Esecuzione principale del flusso dello scenario. Questo scenario orchestra la conversazione tra l'utente, l'API Converse per Amazon Bedrock e uno strumento meteo.

```
/* Before running this JavaScript code example, set up your development environment,
 including your credentials.
This demo illustrates a tool use scenario using Amazon Bedrock's Converse API and a
 weather tool.
The script interacts with a foundation model on Amazon Bedrock to provide weather
 information based on user
input. It uses the Open-Meteo API (https://open-meteo.com) to retrieve current
 weather data for a given location.*/
```

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";
import data from "./questions.json" with { type: "json" };
import toolConfig from "./tool_config.json" with { type: "json" };

const __filename = fileURLToPath(import.meta.url);

const systemPrompt = [
  {
    text:
      "You are a weather assistant that provides current weather data for user-
specified locations using only\n" +
      "the Weather_Tool, which expects latitude and longitude. Infer the coordinates
 from the location yourself.\n" +
      "If the user provides coordinates, infer the approximate location and refer to
 it in your response.\n" +
      "To use the tool, you strictly apply the provided tool specification.\n" +
      "If the user specifies a state, country, or region, infer the locations of
 cities within that state.\n" +
      "\n" +
      "- Explain your step-by-step process, and give brief updates before each step.
\n" +
      "- Only use the Weather_Tool for data. Never guess or make up information. \n"
 +
      "- Repeat the tool use for subsequent requests if necessary.\n" +
      "- If the tool errors, apologize, explain weather is unavailable, and suggest
 other options.\n" +
      "- Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports
 concise. Sparingly use\n" +
      "  emojis where appropriate.\n" +
      "- Only respond to weather queries. Remind off-topic users of your purpose.
 \n" +
```

```
      "- Never claim to search online, access external data, or use tools besides
 Weather_Tool.\n" +
      "- Complete the entire process until you have all required data before sending
 the complete response.",
  },
];
const tools_config = toolConfig;

/// Starts the conversation with the user and handles the interaction with Bedrock.
async function askQuestion(userMessage) {
  // The maximum number of recursive calls allowed in the tool use function.
  // This helps prevent infinite loops and potential performance issues.
  const max_recursions = 5;
  const messages = [
    {
      role: "user",
      content: [{ text: userMessage }],
    },
  ];
  try {
    const response = await SendConversationtoBedrock(messages);
    await ProcessModelResponseAsync(response, messages, max_recursions);
  } catch (error) {
    console.log("error ", error);
  }
}

// Sends the conversation, the system prompt, and the tool spec to Amazon Bedrock,
 and returns the response.
// param "messages" - The conversation history including the next message to send.
// return - The response from Amazon Bedrock.
async function SendConversationtoBedrock(messages) {
  const bedRockRuntimeClient = new BedrockRuntimeClient({
    region: "us-east-1",
  });
  try {
    const modelId = "amazon.nova-lite-v1:0";
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: messages,
        system: systemPrompt,
        toolConfig: tools_config,
      }),
```

```
    );
    return response;
  } catch (caught) {
    if (caught.name === "ModelNotReady") {
      console.log(
        "`${caught.name}` - Model not ready, please wait and try again.",
      );
      throw caught;
    }
    if (caught.name === "BedrockRuntimeException") {
      console.log(
        '`${caught.name}` - "Error occurred while sending Converse request.',
      );
      throw caught;
    }
  }
}

// Processes the response received via Amazon Bedrock and performs the necessary
 actions based on the stop reason.
// param "response" - The model's response returned via Amazon Bedrock.
// param "messages" - The conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function ProcessModelResponseAsync(response, messages, max_recursions) {
  if (max_recursions <= 0) {
    await HandleToolUseAsync(response, messages);
  }
  if (response.stopReason === "tool_use") {
    await HandleToolUseAsync(response, messages, max_recursions - 1);
  }
  if (response.stopReason === "end_turn") {
    const messageToPrint = response.output.message.content[0].text;
    console.log(messageToPrint.replace(/<[^>]+>/g, ""));
  }
}
// Handles the tool use case by invoking the specified tool and sending the tool's
 response back to Bedrock.
// The tool response is appended to the conversation, and the conversation is sent
 back to Amazon Bedrock for further processing.
// param "response" - the model's response containing the tool use request.
// param "messages" - the conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function HandleToolUseAsync(response, messages, max_recursions) {
  const toolResultFinal = [];
```

```
try {
  const output_message = response.output.message;
  messages.push(output_message);
  const toolRequests = output_message.content;
  const toolMessage = toolRequests[0].text;
  console.log(toolMessage.replace(/<[^>]+>/g, ""));
  for (const toolRequest of toolRequests) {
    if (Object.hasOwn(toolRequest, "toolUse")) {
      const toolUse = toolRequest.toolUse;
      const latitude = toolUse.input.latitude;
      const longitude = toolUse.input.longitude;
      const toolUseID = toolUse.toolUseId;
      console.log(
        `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
      );
      if (toolUse.name === "Weather_Tool") {
        try {
          const current_weather = await callWeatherTool(
            longitude,
            latitude,
          ).then((current_weather) => current_weather);
          const currentWeather = current_weather;
          const toolResult = {
            toolResult: {
              toolUseId: toolUseID,
              content: [{ json: currentWeather }],
            },
          };
          toolResultFinal.push(toolResult);
        } catch (err) {
          console.log("An error occurred. ", err);
        }
      }
    }
  }

  const toolResultMessage = {
    role: "user",
    content: toolResultFinal,
  };
  messages.push(toolResultMessage);
  // Send the conversation to Amazon Bedrock
  await ProcessModelResponseAsync(
    await SendConversationtoBedrock(messages),
```

```
      messages,
    );
  } catch (error) {
    console.log("An error occurred. ", error);
  }
}
// Call the Weathertool.
// param = longitude of location
// param = latitude of location
async function callWeatherTool(longitude, latitude) {
  // Open-Meteo API endpoint
  const apiUrl = `https://api.open-meteo.com/v1/forecast?latitude=
${latitude}&longitude=${longitude}&current_weather=true`;

  // Fetch the weather data.
  return fetch(apiUrl)
    .then((response) => {
      return response.json().then((current_weather) => {
        return current_weather;
      });
    })
    .catch((error) => {
      console.error("Error fetching weather data:", error);
    });
}
/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "input",
  default: "",
});

const greet = new ScenarioOutput(
  "greet",
  "Welcome to the Amazon Bedrock Tool Use demo! \n" +
    "This assistant provides current weather information for user-specified
 locations. " +
    "You can ask for weather details by providing the location name or coordinates."
 +
    "Weather information will be provided using a custom Tool and open-meteo API." +
    "For the purposes of this example, we'll use in order the questions in ./
questions.json :\n" +
```

```
    "What's the weather like in Seattle? " +
    "What's the best kind of cat? " +
    "Where is the warmest city in Washington State right now? " +
    "What's the warmest city in California right now?\n" +
    "To exit the program, simply type 'x' and press Enter.\n" +
    "Have fun and experiment with the app by editing the questions in ./
questions.json! " +
    "P.S.: You're not limited to single locations, or even to using English! ",

  { header: true },
);
const displayAskQuestion1 = new ScenarioOutput(
  "displayAskQuestion1",
  "Press enter to ask question number 1 (default is 'What's the weather like in
 Seattle?')",
);

const askQuestion1 = new ScenarioAction(
  "askQuestion1",
  async (/** @type {State} */ state) => {
    const userMessage1 = data.questions["question-1"];
    await askQuestion(userMessage1);
  },
);

const displayAskQuestion2 = new ScenarioOutput(
  "displayAskQuestion2",
  "Press enter to ask question number 2 (default is 'What's the best kind of
 cat?')",
);

const askQuestion2 = new ScenarioAction(
  "askQuestion2",
  async (/** @type {State} */ state) => {
    const userMessage2 = data.questions["question-2"];
    await askQuestion(userMessage2);
  },
);
const displayAskQuestion3 = new ScenarioOutput(
  "displayAskQuestion3",
  "Press enter to ask question number 3 (default is 'Where is the warmest city in
 Washington State right now?')",
);
```

```
const askQuestion3 = new ScenarioAction(
  "askQuestion3",
  async (/** @type {State} */ state) => {
    const userMessage3 = data.questions["question-3"];
    await askQuestion(userMessage3);
  },
);

const displayAskQuestion4 = new ScenarioOutput(
  "displayAskQuestion4",
  "Press enter to ask question number 4 (default is 'What's the warmest city in
 California right now?')",
);

const askQuestion4 = new ScenarioAction(
  "askQuestion4",
  async (/** @type {State} */ state) => {
    const userMessage4 = data.questions["question-4"];
    await askQuestion(userMessage4);
  },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "Thank you for checking out the Amazon Bedrock Tool Use demo. We hope you\n" +
    "learned something new, or got some inspiration for your own apps today!\n" +
    "For more Bedrock examples in different programming languages, have a look at:
\n" +
    "https://docs.aws.amazon.com/bedrock/latest/userguide/
service_code_examples.html",
);

const myScenario = new Scenario("Converse Tool Scenario", [
  greet,
  pressEnter,
  displayAskQuestion1,
  askQuestion1,
  pressEnter,
  displayAskQuestion2,
  askQuestion2,
  pressEnter,
  displayAskQuestion3,
  askQuestion3,
  pressEnter,
```

```
  displayAskQuestion4,
  askQuestion4,
  pressEnter,
  goodbye,
]);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Amazon Nova

Converse

L'esempio di codice seguente mostra come inviare un messaggio di testo ad Amazon Nova utilizzando l'API Converse di Bedrock.

SDK per JavaScript (v3)

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

Invia un messaggio di testo ad Amazon Nova utilizzando l'API Converse di Bedrock.

```
// This example demonstrates how to use the Amazon Nova foundation models to
 generate text.
// It shows how to:
// - Set up the Amazon Bedrock runtime client
// - Create a message
// - Configure and send a request
// - Process the response

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

// Step 1: Create the Amazon Bedrock runtime client
// Credentials will be automatically loaded from the environment.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Step 2: Specify which model to use:
// Available Amazon Nova models and their characteristics:
// - Amazon Nova Micro: Text-only model optimized for lowest latency and cost
// - Amazon Nova Lite:  Fast, low-cost multimodal model for image, video, and text
// - Amazon Nova Pro:   Advanced multimodal model balancing accuracy, speed, and
 cost
//
// For the most current model IDs, see:
// https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
const modelId = "amazon.nova-lite-v1:0";

// Step 3: Create the message
// The message includes the text prompt and specifies that it comes from the user
const inputText =
  "Describe the purpose of a 'hello world' program in one line.";
const message = {
  content: [{ text: inputText }],
  role: ConversationRole.USER,
};

// Step 4: Configure the request
// Optional parameters to control the model's response:
// - maxTokens: maximum number of tokens to generate
// - temperature: randomness (max: 1.0, default: 0.7)
```

```
//   OR
// - topP: diversity of word choice (max: 1.0, default: 0.9)
// Note: Use either temperature OR topP, but not both
const request = {
  modelId,
  messages: [message],
  inferenceConfig: {
    maxTokens: 500, // The maximum response length
    temperature: 0.5, // Using temperature for randomness control
    //topP: 0.9,        // Alternative: use topP instead of temperature
  },
};

// Step 5: Send and process the request
// - Send the request to the model
// - Extract and return the generated text from the response
try {
  const response = await client.send(new ConverseCommand(request));
  console.log(response.output.message.content[0].text);
} catch (error) {
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  throw error;
}
```

Invia una conversazione di messaggi ad Amazon Nova utilizzando l'API Converse di Bedrock con una configurazione dello strumento.

```
// This example demonstrates how to send a conversation of messages to Amazon Nova
 using Bedrock's Converse API with a tool configuration.
// It shows how to:
// - 1. Set up the Amazon Bedrock runtime client
// - 2. Define the parameters required enable Amazon Bedrock to use a tool when
 formulating its response (model ID, user input, system prompt, and the tool spec)
// - 3. Send the request to Amazon Bedrock, and returns the response.
// - 4. Add the tool response to the conversation, and send it back to Amazon
 Bedrock.
// - 5. Publish the response.

import {
  BedrockRuntimeClient,
  ConverseCommand,
```

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

// Step 1: Create the Amazon Bedrock runtime client

// Credentials will be automatically loaded from the environment
const bedRockRuntimeClient = new BedrockRuntimeClient({
  region: "us-east-1",
});

// Step 2. Define the parameters required enable Amazon Bedrock to use a tool when
 formulating its response.

// The Bedrock Model ID.
const modelId = "amazon.nova-lite-v1:0";

// The system prompt to help Amazon Bedrock craft it's response.
const system_prompt = [
  {
    text:
      "You are a music expert that provides the most popular song played on a radio
 station, using only the\n" +
      "the top_song tool, which he call sign for the radio station for which you
 want the most popular song. " +
      "Example calls signs are WZPZ and WKRP. \n" +
      "- Only use the top_song tool. Never guess or make up information. \n" +
      "- If the tool errors, apologize, explain weather is unavailable, and suggest
 other options.\n" +
      "- Only respond to queries about the most popular song played on a radio
 station\n" +
      "Remind off-topic users of your purpose. \n" +
      "- Never claim to search online, access external data, or use tools besides
 the top_song tool.\n",
  },
];
//  The user's question.
const message = [
  {
    role: "user",
    content: [{ text: "What is the most popular song on WZPZ?" }],
  },
];
// The tool specification. In this case, it uses an example schema for
// a tool that gets the most popular song played on a radio station.
const tool_config = {
```

```
  tools: [
    {
      toolSpec: {
        name: "top_song",
        description: "Get the most popular song played on a radio station.",
        inputSchema: {
          json: {
            type: "object",
            properties: {
              sign: {
                type: "string",
                description:
                  "The call sign for the radio station for which you want the most
 popular song. Example calls signs are WZPZ and WKRP.",
              },
            },
            required: ["sign"],
          },
        },
      },
    },
  ],
};

// Helper function to return the song and artist from top_song tool.
async function get_top_song(call_sign) {
  try {
    if (call_sign === "WZPZ") {
      const song = "Elemental Hotel";
      const artist = "8 Storey Hike";
      return { song, artist };
    }
  } catch (error) {
    console.log(`${error.message}`);
  }
}

// 3. Send the request to Amazon Bedrock, and returns the response.
export async function SendConversationtoBedrock(
  modelId,
  message,
  system_prompt,
  tool_config,
) {
```

```
try {
  const response = await bedRockRuntimeClient.send(
    new ConverseCommand({
      modelId: modelId,
      messages: message,
      system: system_prompt,
      toolConfig: tool_config,
    }),
  );
  if (response.stopReason === "tool_use") {
    const toolResultFinal = [];
    try {
      const output_message = response.output.message;
      message.push(output_message);
      const toolRequests = output_message.content;
      const toolMessage = toolRequests[0].text;
      console.log(toolMessage.replace(/<[^>]+>/g, ""));
      for (const toolRequest of toolRequests) {
        if (Object.hasOwn(toolRequest, "toolUse")) {
          const toolUse = toolRequest.toolUse;
          const sign = toolUse.input.sign;
          const toolUseID = toolUse.toolUseId;
          console.log(
            `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
          );
          if (toolUse.name === "top_song") {
            const toolResult = [];
            try {
              const top_song = await get_top_song(toolUse.input.sign).then(
                (top_song) => top_song,
              );
              const toolResult = {
                toolResult: {
                  toolUseId: toolUseID,
                  content: [
                    {
                      json: { song: top_song.song, artist: top_song.artist },
                    },
                  ],
                },
              };
              toolResultFinal.push(toolResult);
            } catch (err) {
              const toolResult = {
```

```
                toolUseId: toolUseID,
                content: [{ json: { text: err.message } }],
                status: "error",
              };
            }
          }
        }
        const toolResultMessage = {
          role: "user",
          content: toolResultFinal,
        };
        // Step 4. Add the tool response to the conversation, and send it back to
Amazon Bedrock.

        message.push(toolResultMessage);
        await SendConversationtoBedrock(
          modelId,
          message,
          system_prompt,
          tool_config,
        );
      } catch (caught) {
        console.error(`${caught.message}`);
        throw caught;
      }
    }

    // 4. Publish the response.
    if (response.stopReason === "end_turn") {
      const finalMessage = response.output.message.content[0].text;
      const messageToPrint = finalMessage.replace(/<[^>]+>/g);
      console.log(messageToPrint.replace(/<[^>]+>/g));
      return messageToPrint;
    }
  } catch (caught) {
    if (caught.name === "ModelNotReady") {
      console.log(
        `${caught.name} - Model not ready, please wait and try again.`,
      );
      throw caught;
    }
    if (caught.name === "BedrockRuntimeException") {
      console.log(
```

```
        `${caught.name} - Error occurred while sending Converse request`,
      );
      throw caught;
    }
  }
}
await SendConversationtoBedrock(modelId, message, system_prompt, tool_config);
```

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK for JavaScript .

ConverseStream

L'esempio di codice seguente mostra come inviare un messaggio di testo ad Amazon Nova utilizzando l'API Converse di Bedrock ed elaborare il flusso di risposta in tempo reale.

SDK per JavaScript (v3)

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

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

```
// This example demonstrates how to use the Amazon Nova foundation models
// to generate streaming text responses.
// It shows how to:
// - Set up the Amazon Bedrock runtime client
// - Create a message
// - Configure a streaming request
// - Process the streaming response

import {
  BedrockRuntimeClient,
  ConversationRole,
  ConverseStreamCommand,
```

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

// Step 1: Create the Amazon Bedrock runtime client
// Credentials will be automatically loaded from the environment
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Step 2: Specify which model to use
// Available Amazon Nova models and their characteristics:
// - Amazon Nova Micro: Text-only model optimized for lowest latency and cost
// - Amazon Nova Lite:  Fast, low-cost multimodal model for image, video, and text
// - Amazon Nova Pro:   Advanced multimodal model balancing accuracy, speed, and
 cost
//
// For the most current model IDs, see:
// https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
const modelId = "amazon.nova-lite-v1:0";

// Step 3: Create the message
// The message includes the text prompt and specifies that it comes from the user
const inputText =
  "Describe the purpose of a 'hello world' program in one paragraph";
const message = {
  content: [{ text: inputText }],
  role: ConversationRole.USER,
};

// Step 4: Configure the streaming request
// Optional parameters to control the model's response:
// - maxTokens: maximum number of tokens to generate
// - temperature: randomness (max: 1.0, default: 0.7)
//   OR
// - topP: diversity of word choice (max: 1.0, default: 0.9)
// Note: Use either temperature OR topP, but not both
const request = {
  modelId,
  messages: [message],
  inferenceConfig: {
    maxTokens: 500, // The maximum response length
    temperature: 0.5, // Using temperature for randomness control
    //topP: 0.9,        // Alternative: use topP instead of temperature
  },
};

// Step 5: Send and process the streaming request
```

```
// - Send the request to the model
// - Process each chunk of the streaming response
try {
  const response = await client.send(new ConverseStreamCommand(request));

  for await (const chunk of response.stream) {
    if (chunk.contentBlockDelta) {
      // Print each text chunk as it arrives
      process.stdout.write(chunk.contentBlockDelta.delta?.text || "");
    }
  }
} catch (error) {
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  process.exitCode = 1;
}
```

- Per i dettagli sull'API, ConverseStreamconsulta AWS SDK for JavaScript API Reference.

Scenario: utilizzo dello strumento con l'API Converse

Il seguente esempio di codice mostra come creare un'interazione tipica tra un'applicazione, un modello di intelligenza artificiale generativa e strumenti connessi o come APIs mediare le interazioni tra l'IA e il mondo esterno. Viene utilizzato l'esempio di collegamento di un'API esterna per il meteo al modello di IA in modo che vengano fornite informazioni meteorologiche in tempo reale in base sull'input dell'utente.

SDK per (v3 JavaScript)

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

Esecuzione principale del flusso dello scenario. Questo scenario orchestra la conversazione tra l'utente, l'API Converse per Amazon Bedrock e uno strumento meteo.

```
/* Before running this JavaScript code example, set up your development environment,
 including your credentials.
```

```
This demo illustrates a tool use scenario using Amazon Bedrock's Converse API and a
 weather tool.
The script interacts with a foundation model on Amazon Bedrock to provide weather
 information based on user
input. It uses the Open-Meteo API (https://open-meteo.com) to retrieve current
 weather data for a given location.*/

import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";
import data from "./questions.json" with { type: "json" };
import toolConfig from "./tool_config.json" with { type: "json" };

const __filename = fileURLToPath(import.meta.url);

const systemPrompt = [
  {
    text:
      "You are a weather assistant that provides current weather data for user-
specified locations using only\n" +
      "the Weather_Tool, which expects latitude and longitude. Infer the coordinates
 from the location yourself.\n" +
      "If the user provides coordinates, infer the approximate location and refer to
 it in your response.\n" +
      "To use the tool, you strictly apply the provided tool specification.\n" +
      "If the user specifies a state, country, or region, infer the locations of
 cities within that state.\n" +
      "\n" +
      "- Explain your step-by-step process, and give brief updates before each step.
\n" +
      "- Only use the Weather_Tool for data. Never guess or make up information. \n"
 +
      "- Repeat the tool use for subsequent requests if necessary.\n" +
```

```
      "- If the tool errors, apologize, explain weather is unavailable, and suggest
 other options.\n" +
      "- Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports
 concise. Sparingly use\n" +
      "  emojis where appropriate.\n" +
      "- Only respond to weather queries. Remind off-topic users of your purpose.
 \n" +
      "- Never claim to search online, access external data, or use tools besides
 Weather_Tool.\n" +
      "- Complete the entire process until you have all required data before sending
 the complete response.",
  },
];
const tools_config = toolConfig;

/// Starts the conversation with the user and handles the interaction with Bedrock.
async function askQuestion(userMessage) {
  // The maximum number of recursive calls allowed in the tool use function.
  // This helps prevent infinite loops and potential performance issues.
  const max_recursions = 5;
  const messages = [
    {
      role: "user",
      content: [{ text: userMessage }],
    },
  ];
  try {
    const response = await SendConversationtoBedrock(messages);
    await ProcessModelResponseAsync(response, messages, max_recursions);
  } catch (error) {
    console.log("error ", error);
  }
}

// Sends the conversation, the system prompt, and the tool spec to Amazon Bedrock,
 and returns the response.
// param "messages" - The conversation history including the next message to send.
// return - The response from Amazon Bedrock.
async function SendConversationtoBedrock(messages) {
  const bedRockRuntimeClient = new BedrockRuntimeClient({
    region: "us-east-1",
  });
  try {
    const modelId = "amazon.nova-lite-v1:0";
```

```
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: messages,
        system: systemPrompt,
        toolConfig: tools_config,
      }),
    );
    return response;
  } catch (caught) {
    if (caught.name === "ModelNotReady") {
      console.log(
        "`${caught.name}` - Model not ready, please wait and try again.",
      );
      throw caught;
    }
    if (caught.name === "BedrockRuntimeException") {
      console.log(
        '`${caught.name}` - "Error occurred while sending Converse request.',
      );
      throw caught;
    }
  }
}

// Processes the response received via Amazon Bedrock and performs the necessary
 actions based on the stop reason.
// param "response" - The model's response returned via Amazon Bedrock.
// param "messages" - The conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function ProcessModelResponseAsync(response, messages, max_recursions) {
  if (max_recursions <= 0) {
    await HandleToolUseAsync(response, messages);
  }
  if (response.stopReason === "tool_use") {
    await HandleToolUseAsync(response, messages, max_recursions - 1);
  }
  if (response.stopReason === "end_turn") {
    const messageToPrint = response.output.message.content[0].text;
    console.log(messageToPrint.replace(/<[^>]+>/g, ""));
  }
}
// Handles the tool use case by invoking the specified tool and sending the tool's
 response back to Bedrock.
```

```
// The tool response is appended to the conversation, and the conversation is sent
 back to Amazon Bedrock for further processing.
// param "response" - the model's response containing the tool use request.
// param "messages" - the conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function HandleToolUseAsync(response, messages, max_recursions) {
  const toolResultFinal = [];
  try {
    const output_message = response.output.message;
    messages.push(output_message);
    const toolRequests = output_message.content;
    const toolMessage = toolRequests[0].text;
    console.log(toolMessage.replace(/<[^>]+>/g, ""));
    for (const toolRequest of toolRequests) {
      if (Object.hasOwn(toolRequest, "toolUse")) {
        const toolUse = toolRequest.toolUse;
        const latitude = toolUse.input.latitude;
        const longitude = toolUse.input.longitude;
        const toolUseID = toolUse.toolUseId;
        console.log(
          `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
        );
        if (toolUse.name === "Weather_Tool") {
          try {
            const current_weather = await callWeatherTool(
              longitude,
              latitude,
            ).then((current_weather) => current_weather);
            const currentWeather = current_weather;
            const toolResult = {
              toolResult: {
                toolUseId: toolUseID,
                content: [{ json: currentWeather }],
              },
            };
            toolResultFinal.push(toolResult);
          } catch (err) {
            console.log("An error occurred. ", err);
          }
        }
      }
    }

    const toolResultMessage = {
```

```
      role: "user",
      content: toolResultFinal,
    };
    messages.push(toolResultMessage);
    // Send the conversation to Amazon Bedrock
    await ProcessModelResponseAsync(
      await SendConversationtoBedrock(messages),
      messages,
    );
  } catch (error) {
    console.log("An error occurred. ", error);
  }
}
// Call the Weathertool.
// param = longitude of location
// param = latitude of location
async function callWeatherTool(longitude, latitude) {
  // Open-Meteo API endpoint
  const apiUrl = `https://api.open-meteo.com/v1/forecast?latitude=
${latitude}&longitude=${longitude}&current_weather=true`;

  // Fetch the weather data.
  return fetch(apiUrl)
    .then((response) => {
      return response.json().then((current_weather) => {
        return current_weather;
      });
    })
    .catch((error) => {
      console.error("Error fetching weather data:", error);
    });
}
/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "input",
  default: "",
});

const greet = new ScenarioOutput(
  "greet",
  "Welcome to the Amazon Bedrock Tool Use demo! \n" +
```

```
    "This assistant provides current weather information for user-specified
 locations. " +
    "You can ask for weather details by providing the location name or coordinates."
 +
    "Weather information will be provided using a custom Tool and open-meteo API." +
    "For the purposes of this example, we'll use in order the questions in ./
questions.json :\n" +
    "What's the weather like in Seattle? " +
    "What's the best kind of cat? " +
    "Where is the warmest city in Washington State right now? " +
    "What's the warmest city in California right now?\n" +
    "To exit the program, simply type 'x' and press Enter.\n" +
    "Have fun and experiment with the app by editing the questions in ./
questions.json! " +
    "P.S.: You're not limited to single locations, or even to using English! ",

  { header: true },
);
const displayAskQuestion1 = new ScenarioOutput(
  "displayAskQuestion1",
  "Press enter to ask question number 1 (default is 'What's the weather like in
 Seattle?')",
);

const askQuestion1 = new ScenarioAction(
  "askQuestion1",
  async (/** @type {State} */ state) => {
    const userMessage1 = data.questions["question-1"];
    await askQuestion(userMessage1);
  },
);

const displayAskQuestion2 = new ScenarioOutput(
  "displayAskQuestion2",
  "Press enter to ask question number 2 (default is 'What's the best kind of
 cat?')",
);

const askQuestion2 = new ScenarioAction(
  "askQuestion2",
  async (/** @type {State} */ state) => {
    const userMessage2 = data.questions["question-2"];
    await askQuestion(userMessage2);
  },
```

```
);
const displayAskQuestion3 = new ScenarioOutput(
  "displayAskQuestion3",
  "Press enter to ask question number 3 (default is 'Where is the warmest city in
 Washington State right now?')",
);

const askQuestion3 = new ScenarioAction(
  "askQuestion3",
  async (/** @type {State} */ state) => {
    const userMessage3 = data.questions["question-3"];
    await askQuestion(userMessage3);
  },
);

const displayAskQuestion4 = new ScenarioOutput(
  "displayAskQuestion4",
  "Press enter to ask question number 4 (default is 'What's the warmest city in
 California right now?')",
);

const askQuestion4 = new ScenarioAction(
  "askQuestion4",
  async (/** @type {State} */ state) => {
    const userMessage4 = data.questions["question-4"];
    await askQuestion(userMessage4);
  },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "Thank you for checking out the Amazon Bedrock Tool Use demo. We hope you\n" +
    "learned something new, or got some inspiration for your own apps today!\n" +
    "For more Bedrock examples in different programming languages, have a look at:
\n" +
    "https://docs.aws.amazon.com/bedrock/latest/userguide/
service_code_examples.html",
);

const myScenario = new Scenario("Converse Tool Scenario", [
  greet,
  pressEnter,
  displayAskQuestion1,
  askQuestion1,
```

```
  pressEnter,
  displayAskQuestion2,
  askQuestion2,
  pressEnter,
  displayAskQuestion3,
  askQuestion3,
  pressEnter,
  displayAskQuestion4,
  askQuestion4,
  pressEnter,
  goodbye,
]);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Amazon Nova Canvas

InvokeModel

L'esempio di codice seguente mostra come invocare Amazon Nova Canvas in Amazon Bedrock per generare un'immagine.

SDK per JavaScript (v3)

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

Crea un'immagine con Amazon Nova Canvas.

```
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
import { saveImage } from "../../utils/image-creation.js";
import { fileURLToPath } from "node:url";

/**
 * This example demonstrates how to use Amazon Nova Canvas to generate images.
 * It shows how to:
 * - Set up the Amazon Bedrock runtime client
 * - Configure the image generation parameters
 * - Send a request to generate an image
 * - Process the response and handle the generated image
 *
 * @returns {Promise<string>} Base64-encoded image data
 */
export const invokeModel = async () => {
  // Step 1: Create the Amazon Bedrock runtime client
  // Credentials will be automatically loaded from the environment
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Step 2: Specify which model to use
  // For the latest available models, see:
  // https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
  const modelId = "amazon.nova-canvas-v1:0";

  // Step 3: Configure the request payload
  // First, set the main parameters:
  // - prompt: Text description of the image to generate
  // - seed: Random number for reproducible generation (0 to 858,993,459)
  const prompt = "A stylized picture of a cute old steampunk robot";
```

```
  const seed = Math.floor(Math.random() * 858993460);

  // Then, create the payload using the following structure:
  // - taskType: TEXT_IMAGE (specifies text-to-image generation)
  // - textToImageParams: Contains the text prompt
  // - imageGenerationConfig: Contains optional generation settings (seed, quality,
 etc.)
  // For a list of available request parameters, see:
  // https://docs.aws.amazon.com/nova/latest/userguide/image-gen-req-resp-
structure.html
  const payload = {
    taskType: "TEXT_IMAGE",
    textToImageParams: {
      text: prompt,
    },
    imageGenerationConfig: {
      seed,
      quality: "standard",
    },
  };

  // Step 4: Send and process the request
  // - Embed the payload in a request object
  // - Send the request to the model
  // - Extract and return the generated image data from the response
  try {
    const request = {
      modelId,
      body: JSON.stringify(payload),
    };
    const response = await client.send(new InvokeModelCommand(request));

    const decodedResponseBody = new TextDecoder().decode(response.body);
    // The response includes an array of base64-encoded PNG images
    /** @type {{images: string[]}} */
    const responseBody = JSON.parse(decodedResponseBody);
    return responseBody.images[0]; // Base64-encoded image data
  } catch (error) {
    console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
    throw error;
  }
};

// If run directly, execute the example and save the generated image
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("Generating image. This may take a few seconds...");
  invokeModel()
    .then(async (imageData) => {
      const imagePath = await saveImage(imageData, "nova-canvas");
      // Example path: javascriptv3/example_code/bedrock-runtime/output/nova-canvas/
image-01.png
      console.log(`Image saved to: ${imagePath}`);
    })
    .catch((error) => {
      console.error("Execution failed:", error);
      process.exitCode = 1;
    });
}
```

- Per i dettagli sull'API, InvokeModelconsulta AWS SDK for JavaScript API Reference.

Anthropic Claude

Converse

L'esempio di codice seguente mostra come inviare un messaggio di testo ad Anthropic Claude utilizzando l'API Converse di Bedrock.

SDK per JavaScript (v3)

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

Invia un messaggio di testo ad Anthropic Claude utilizzando l'API Converse di Bedrock.

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

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

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

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

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

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

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

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

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK for JavaScript .

ConverseStream

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

SDK per JavaScript (v3)

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

Invia un messaggio di testo ad Anthropic Claude utilizzando l'API Converse di Bedrock 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 dettagli sull'API, ConverseStreamconsulta AWS SDK for JavaScript API Reference.

InvokeModel

L'esempio di codice seguente mostra come inviare un messaggio di testo a Anthropic Claude, utilizzando l'API Invoke Model.

SDK per JavaScript (v3)

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

Utilizza l'API Invoke Model per inviare un messaggio di testo.

```
import { fileURLToPath } from "node: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 i dettagli sull'API, InvokeModelconsulta AWS SDK for JavaScript API Reference.

InvokeModelWithResponseStream

L'esempio di codice seguente mostra come inviare un messaggio di testo ai modelli Anthropic Claude utilizzando l'API Invoke Model e stampare il flusso di risposta.

SDK per JavaScript (v3)

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

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
import { fileURLToPath } from "node: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 i dettagli sull'API, InvokeModelWithResponseStreamconsulta AWS SDK for JavaScript API Reference.

Cohere Command

Converse

L'esempio di codice seguente mostra come inviare un messaggio di testo a Cohere Command utilizzando l'API Converse di Bedrock.

SDK per JavaScript (v3)

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

Invia un messaggio di testo a Cohere Command, utilizzando l'API Converse di Bedrock.

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

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

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

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

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

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

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

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

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK for JavaScript .

ConverseStream

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

SDK per JavaScript (v3)

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

Invia un messaggio di testo a Cohere Command utilizzando l'API Converse di Bedrock 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 dettagli sull'API, ConverseStreamconsulta AWS SDK for JavaScript API Reference.

Meta Llama

Converse

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

SDK per JavaScript (v3)

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

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

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

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

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

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

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

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

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

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

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK for JavaScript .

ConverseStream

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

SDK per JavaScript (v3)

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

Invia un messaggio di testo a Meta Llama utilizzando l'API Converse di Bedrock 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 dettagli sull'API, ConverseStreamconsulta AWS SDK for JavaScript API Reference.

InvokeModel

L'esempio di codice seguente mostra come inviare un messaggio di testo a Meta Llama utilizzando l'API Invoke Model.

SDK per JavaScript (v3)

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

Utilizza l'API Invoke Model 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 70B Instruct.
const modelId = "meta.llama3-70b-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 i dettagli sull'API, InvokeModelconsulta AWS SDK for JavaScript API Reference.

InvokeModelWithResponseStream

L'esempio di codice seguente mostra come inviare un messaggio di testo a Meta Llama utilizzando l'API Invoke Model e stampare il flusso di risposta.

SDK per JavaScript (v3)

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

Utilizza l'API Invoke Model 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 70B Instruct.
const modelId = "meta.llama3-70b-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 i dettagli sull'API, InvokeModelWithResponseStreamconsulta AWS SDK for JavaScript API Reference.

Mistral AI

Converse

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

SDK per JavaScript (v3)

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

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

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

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

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

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

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

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

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

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

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

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK for JavaScript .

ConverseStream

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

SDK per JavaScript (v3)

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

Invia un messaggio di testo a Mistral utilizzando l'API Converse di Bedrock 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 dettagli sull'API, ConverseStreamconsulta AWS SDK for JavaScript API Reference.

InvokeModel

L'esempio di codice seguente mostra come inviare un messaggio di testo ai modelli Mistral utilizzando l'API Invoke Model.

SDK per JavaScript (v3)

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

Utilizza l'API Invoke Model per inviare un messaggio di testo.

```
import { fileURLToPath } from "node: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 i dettagli sull'API, InvokeModelconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon Bedrock Agents 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 Bedrock Agents.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Azioni

Nozioni di base

Hello Amazon Bedrock Agents

L'esempio di codice seguente mostra come iniziare a utilizzare Agent per Amazon Bedrock.

SDK per (v3 JavaScript)

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

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

import {
  BedrockAgentClient,
  GetAgentCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * @typedef {Object} AgentSummary
 */

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 region.
 * It then retrieves a list of existing agents using the streamlined paginator
 approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
 * - Use of the Bedrock Agents client to initialize and communicate with the AWS
 service.
 * - Listing resources in a paginated response pattern.
 * - Accessing an individual resource using a command object.
 *
 * @returns {Promise<void>} A promise that resolves when the function has completed
 execution.
 */
export const main = async () => {
  const region = "us-east-1";

  console.log("=".repeat(68));

  console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
  const client = new BedrockAgentClient({ region });

  console.log("Retrieving the list of existing agents...");
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});
```

```
  /** @type {AgentSummary[]} */
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  console.log(`Found ${agentSummaries.length} agents in ${region}.`);

  if (agentSummaries.length > 0) {
    for (const agentSummary of agentSummaries) {
      const agentId = agentSummary.agentId;
      console.log("=".repeat(68));
      console.log(`Retrieving agent with ID: ${agentId}:`);
      console.log("-".repeat(68));

      const command = new GetAgentCommand({ agentId });
      const response = await client.send(command);
      const agent = response.agent;

      console.log(` Name: ${agent.agentName}`);
      console.log(` Status: ${agent.agentStatus}`);
      console.log(` ARN: ${agent.agentArn}`);
      console.log(` Foundation model: ${agent.foundationModel}`);
    }
  }
  console.log("=".repeat(68));
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - GetAgent
  - ListAgents

Azioni

`CreateAgent`

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

SDK per JavaScript (v3)

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

Crea un agente.

```
import { fileURLToPath } from "node:url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  CreateAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Creates an Amazon Bedrock Agent.
 *
 * @param {string} agentName - A name for the agent that you create.
 * @param {string} foundationModel - The foundation model to be used by the agent
 you create.
 * @param {string} agentResourceRoleArn - The ARN of the IAM role with permissions
 required by the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing details of the created agent.
 */
export const createAgent = async (
  agentName,
  foundationModel,
  agentResourceRoleArn,
  region = "us-east-1",
) => {
```

```
  const client = new BedrockAgentClient({ region });

  const command = new CreateAgentCommand({
    agentName,
    foundationModel,
    agentResourceRoleArn,
  });
  const response = await client.send(command);

  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentName and accountId, and roleName with a
 unique name for the new agent,
  // the id of your AWS account, and the name of an existing execution role that the
 agent can use inside your account.
  // For foundationModel, specify the desired model. Ensure to remove the brackets
 '[]' before adding your data.

  // A string (max 100 chars) that can include letters, numbers, dashes '-', and
 underscores '_'.
  const agentName = "[your-bedrock-agent-name]";

  // Your AWS account id.
  const accountId = "[123456789012]";

  // The name of the agent's execution role. It must be prefixed by
 `AmazonBedrockExecutionRoleForAgents_`.
  const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

  // The ARN for the agent's execution role.
  // Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
  const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

  // Specify the model for the agent. Change if a different model is preferred.
  const foundationModel = "anthropic.claude-v2";

  // Check for unresolved placeholders in agentName and roleArn.
  checkForPlaceholders([agentName, roleArn]);

  console.log("Creating a new agent...");
```

```
  const agent = await createAgent(agentName, foundationModel, roleArn);
  console.log(agent);
}
```

- Per i dettagli sull'API, CreateAgentconsulta AWS SDK for JavaScript API Reference.

`DeleteAgent`

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

SDK per JavaScript (v3)

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

Elimina un agente.

```
import { fileURLToPath } from "node:url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent to delete.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-
agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
 and some additional metadata.
 */
export const deleteAgent = (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });
  const command = new DeleteAgentCommand({ agentId });
```

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

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets (`[]`) before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Deleting agent with ID ${agentId}...`);

  const response = await deleteAgent(agentId);
  console.log(response);
}
```

- Per i dettagli sull'API, DeleteAgentconsulta AWS SDK for JavaScript API Reference.

`GetAgent`

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

SDK per JavaScript (v3)

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

Ottiene un agente.

```
import { fileURLToPath } from "node:url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
```

```
  BedrockAgentClient,
  GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing the agent details.
 */
export const getAgent = async (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const command = new GetAgentCommand({ agentId });
  const response = await client.send(command);
  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Retrieving agent with ID ${agentId}...`);

  const agent = await getAgent(agentId);
  console.log(agent);
}
```

- Per i dettagli sull'API, GetAgentconsulta AWS SDK for JavaScript API Reference.

`ListAgentActionGroups`

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

SDK per JavaScript (v3)

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

Elenca i gruppi di azioni per un agente.

```
import { fileURLToPath } from "node:url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  ListAgentActionGroupsCommand,
  paginateListAgentActionGroups,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  // Create a paginator configuration
  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
```

```
  };

  const params = { agentId, agentVersion };

  const pages = paginateListAgentActionGroups(paginatorConfig, params);

  // Paginate until there are no more results
  const actionGroupSummaries = [];
  for await (const page of pages) {
    actionGroupSummaries.push(...page.actionGroupSummaries);
  }

  return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const actionGroupSummaries = [];
  do {
    const command = new ListAgentActionGroupsCommand({
      agentId,
      agentVersion,
      nextToken,
```

```
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
    const response = await client.send(command);

    for (const actionGroup of response.actionGroupSummaries || []) {
      actionGroupSummaries.push(actionGroup);
    }

    nextToken = response.nextToken;
  } while (nextToken);

  return actionGroupSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId and agentVersion with an existing agent's
 id and version.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
 'DRAFT').
  const agentVersion = "[DRAFT]";

  // Check for unresolved placeholders in agentId and agentVersion.
  checkForPlaceholders([agentId, agentVersion]);

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using ListAgentActionGroupsCommand:",
  );

  for (const actionGroup of await listAgentActionGroupsWithCommandObject(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }
```

```
  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using the paginateListAgents function:",
  );
  for (const actionGroup of await listAgentActionGroupsWithPaginator(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }
}
```

- Per i dettagli sull'API, ListAgentActionGroupsconsulta AWS SDK for JavaScript API Reference.

`ListAgents`

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

SDK per JavaScript (v3)

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

Elenca gli agenti appartenenti a un account.

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

import {
  BedrockAgentClient,
  ListAgentsCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 function.
 *
```

```
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 ListAgentsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const agentSummaries = [];
```

```
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
    const paginatedResponse = await client.send(command);

    agentSummaries.push(...(paginatedResponse.agentSummaries || []));

    nextToken = paginatedResponse.nextToken;
  } while (nextToken);

  return agentSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
  console.log("Listing agents using ListAgentsCommand:");
  for (const agent of await listAgentsWithCommandObject()) {
    console.log(agent);
  }

  console.log("=".repeat(68));
  console.log("Listing agents using the paginateListAgents function:");
  for (const agent of await listAgentsWithPaginator()) {
    console.log(agent);
  }
}
```

- Per i dettagli sull'API, ListAgentsconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon Bedrock Agents 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 Amazon Bedrock Agents Runtime.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`InvokeAgent`

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

SDK per JavaScript (v3)

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

```
import {
  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 (const 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 "node:url";
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- Per i dettagli sull'API, InvokeAgentconsulta AWS SDK for JavaScript API Reference.

`InvokeFlow`

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

SDK per JavaScript (v3)

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

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

import {
  BedrockAgentRuntimeClient,
  InvokeFlowCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * Invokes an alias of a flow to run the inputs that you specify and return
 * the output of each node as a stream.
 *
 * @param {{
 *  flowIdentifier: string,
 *  flowAliasIdentifier: string,
 *  prompt?: string,
 *  region?: string
 * }} options
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").FlowNodeOutput>} An
 object containing information about the output from flow invocation.
 */
export const invokeBedrockFlow = async ({
```

```
  flowIdentifier,
  flowAliasIdentifier,
  prompt = "Hi, how are you?",
  region = "us-east-1",
}) => {
  const client = new BedrockAgentRuntimeClient({ region });

  const command = new InvokeFlowCommand({
    flowIdentifier,
    flowAliasIdentifier,
    inputs: [
      {
        content: {
          document: prompt,
        },
        nodeName: "FlowInputNode",
        nodeOutputName: "document",
      },
    ],
  });

  let flowResponse = {};
  const response = await client.send(command);

  for await (const chunkEvent of response.responseStream) {
    const { flowOutputEvent, flowCompletionEvent } = chunkEvent;

    if (flowOutputEvent) {
      flowResponse = { ...flowResponse, ...flowOutputEvent };
      console.log("Flow output event:", flowOutputEvent);
    } else if (flowCompletionEvent) {
      flowResponse = { ...flowResponse, ...flowCompletionEvent };
      console.log("Flow completion event:", flowCompletionEvent);
    }
  }

  return flowResponse;
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
```

```
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    flowIdentifier: {
      type: "string",
      required: true,
    },
    flowAliasIdentifier: {
      type: "string",
      required: true,
    },
    prompt: {
      type: "string",
    },
    region: {
      type: "string",
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    invokeBedrockFlow(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per i dettagli sull'API, InvokeFlowconsulta AWS SDK for JavaScript API Reference.

CloudWatch esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. CloudWatch

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`DeleteAlarms`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};
```

```
export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteAlarmsconsulta AWS SDK for JavaScript API Reference.

`DescribeAlarmsForMetric`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
```

```
  }
};

export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DescribeAlarmsForMetricconsulta AWS SDK for JavaScript API Reference.

`DisableAlarmActions`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
```

```
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DisableAlarmActionsconsulta AWS SDK for JavaScript API Reference.

`EnableAlarmActions`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });
```

```
  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, EnableAlarmActionsconsulta AWS SDK for JavaScript API Reference.

`ListMetrics`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import {
  CloudWatchServiceException,
  ListMetricsCommand,
} from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";
```

```
export const main = async () => {
  // 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",
  });

  try {
    const response = await client.send(command);
    console.log(`Metrics count: ${response.Metrics?.length}`);
    return response;
  } catch (caught) {
    if (caught instanceof CloudWatchServiceException) {
      console.error(`Error from CloudWatch. ${caught.name}: ${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListMetricsconsulta AWS SDK for JavaScript API Reference.

`PutMetricAlarm`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
 the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};
```

```
export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutMetricAlarmconsulta AWS SDK for JavaScript API Reference.

`PutMetricData`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
```

```
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutMetricDataconsulta AWS SDK for JavaScript API Reference.

CloudWatch Esempi di eventi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with CloudWatch Events.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`PutEvents`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
        Source: "my.app",

        // The name of the event that is being sent.
        DetailType: "My Custom Event",
```

```
        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutEventsconsulta AWS SDK for JavaScript API Reference.

`PutRule`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
```

```
import { client } from "../libs/client.js";

const run = async () => {
  // Request parameters for PutRule.
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/
API_PutRule.html#API_PutRule_RequestParameters
  const command = new PutRuleCommand({
    Name: process.env.CLOUDWATCH_EVENTS_RULE,

    // The event pattern for the rule.
    //  Example: {"source": ["my.app"]}
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,

    // The state of the rule. Valid values: ENABLED, DISABLED
    State: "ENABLED",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutRuleconsulta AWS SDK for JavaScript API Reference.

`PutTargets`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutTargetsCommand({
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,

    // The targets to add to the rule.
    Targets: [
      {
        Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
        // The ID of the target. Choose a unique ID for each target.
        Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Creare il client in un modulo separato ed esportarlo.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";
```

```
export const client = new CloudWatchEventsClient({});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutTargetsconsulta AWS SDK for JavaScript API Reference.

CloudWatch Registra esempi utilizzando SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with Logs. CloudWatch

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni
- Scenari

Azioni

`CreateLogGroup`

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

SDK per JavaScript (v3)

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

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Per i dettagli sull'API, CreateLogGroupconsulta AWS SDK for JavaScript API Reference.

`DeleteLogGroup`

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

SDK per JavaScript (v3)

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

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });
```

```
  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Per i dettagli sull'API, DeleteLogGroupconsulta AWS SDK for JavaScript API Reference.

`DeleteSubscriptionFilter`

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

SDK per JavaScript (v3)

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

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};
```

```
export default run();
```

- Per i dettagli sull'API, DeleteSubscriptionFilterconsulta AWS SDK for JavaScript API Reference.

`DescribeLogGroups`

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

SDK per JavaScript (v3)

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

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups?.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }

  console.log(logGroups);
  return logGroups;
};
```

- Per i dettagli sull'API, DescribeLogGroupsconsulta AWS SDK for JavaScript API Reference.

`DescribeSubscriptionFilters`

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

SDK per JavaScript (v3)

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

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-
logs";
import { client } from "../libs/client.js";

const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Per i dettagli sull'API, DescribeSubscriptionFiltersconsulta AWS SDK for JavaScript API Reference.

`GetQueryResults`

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

SDK per JavaScript (v3)

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

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

- Per i dettagli sull'API, GetQueryResultsconsulta AWS SDK for JavaScript API Reference.

`PutSubscriptionFilter`

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

SDK per JavaScript (v3)

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

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
    // An ARN of a same-account Kinesis stream, Kinesis Firehose
    // delivery stream, or Lambda function.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
SubscriptionFilters.html
```

```
    destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,

    // A name for the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

    // A filter pattern for subscribing to a filtered stream of log events.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
FilterAndPatternSyntax.html
    filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

    // The name of the log group. Messages in this group matching the filter pattern
    // will be sent to the destination ARN.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Per i dettagli sull'API, PutSubscriptionFilterconsulta AWS SDK for JavaScript API Reference.

`StartLiveTail`

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

SDK per JavaScript (v3)

Includere i file richiesti.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-
cloudwatch-logs";
```

Gestisce gli eventi della sessione 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);
    }
```

Arresta la sessione Live Tail dopo un determinato periodo di tempo.

```
    /* Set a timeout to close the client. This will stop the Live Tail session. */
    setTimeout(function() {
```

```
        console.log("Client timeout");
        client.destroy();
      }, 10000);
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API StartLiveTailReference.

`StartQuery`

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

SDK per JavaScript (v3)

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

```
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
```

```
        // before the log group was created.
        throw new DateOutOfBoundsError(message);
      }

      throw err;
    }
  }
```

- Per i dettagli sull'API, StartQueryconsulta AWS SDK for JavaScript API Reference.

Scenari

Eseguire una query di grandi dimensioni

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

SDK per JavaScript (v3)

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

Questo è il punto di ingresso.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}
```

```
const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
  logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
  dateRange: [
    new Date(Number.parseInt(process.env.QUERY_START_DATE)),
    new Date(Number.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 suddivide le query 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 [];
      }
      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 informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - GetQueryResults
  - StartQuery

Utilizzo degli eventi pianificati per invocare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

SDK per JavaScript (v3)

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

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 azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`CreateProject`

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

SDK per JavaScript (v3)

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

Crea un progetto.

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
} from "@aws-sdk/client-codebuild";
```

```
// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});

  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
        location: buildOutputBucket,
      },
      // Information about the build environment. The combination of "computeType"
 and "type" determines the
      // requirements for the environment such as CPU, memory, and disk space.
      environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
compute-types.html
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
      },
      name: projectName,
      // A role ARN with permission to create a CodeBuild project, write to the
 artifact location, and write CloudWatch logs.
      serviceRole: roleArn,
      source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be built.
        location: githubUrl,
      },
    }),
  );
  console.log(response);
```

```
//   {
//     '$metadata': {
//       httpStatusCode: 200,
//       requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
//       extendedRequestId: undefined,
//       cfId: undefined,
//       attempts: 1,
//       totalRetryDelay: 0
//     },
//     project: {
//       arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
//       artifacts: {
//         encryptionDisabled: false,
//         location: 'xxxxxx-xxxxxxx-xxxxxx',
//         name: 'MyCodeBuilder',
//         namespaceType: 'NONE',
//         packaging: 'NONE',
//         type: 'S3'
//       },
//       badge: { badgeEnabled: false },
//       cache: { type: 'NO_CACHE' },
//       created: 2023-08-18T14:46:48.979Z,
//       encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
//       environment: {
//         computeType: 'BUILD_GENERAL1_SMALL',
//         environmentVariables: [],
//         image: 'aws/codebuild/standard:7.0',
//         imagePullCredentialsType: 'CODEBUILD',
//         privilegedMode: false,
//         type: 'LINUX_CONTAINER'
//       },
//       lastModified: 2023-08-18T14:46:48.979Z,
//       name: 'MyCodeBuilder',
//       projectVisibility: 'PRIVATE',
//       queuedTimeoutInMinutes: 480,
//       serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
//       source: {
//         insecureSsl: false,
//         location: 'https://...',
//         reportBuildStatus: false,
//         type: 'GITHUB'
//       },
//       timeoutInMinutes: 60
//     }
```

```
  //   }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateProjectconsulta AWS SDK for JavaScript API Reference.

Esempi di identità di Amazon Cognito 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 Cognito Identity.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Scenari

Scenari

Creazione di un'applicazione Amazon Textract explorer

L'esempio di codice seguente mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDK per (v3 JavaScript)

> 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 SQS) sottoscritta a un argomento 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
- Simple Storage Service (Amazon S3)
- Amazon SNS
- Amazon SQS
- Amazon Textract

Esempi di Amazon Cognito Identity Provider 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 Cognito Identity Provider.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Azioni
- Scenari

Nozioni di base

Ciao Amazon Cognito

L'esempio di codice seguente mostra come iniziare a utilizzare Amazon Cognito.

SDK per (v3 JavaScript)

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

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- Per i dettagli sull'API, ListUserPoolsconsulta AWS SDK for JavaScript API Reference.

Azioni

`AdminGetUser`

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

SDK per JavaScript (v3)

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

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

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

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

- Per i dettagli sull'API, AdminGetUserconsulta AWS SDK for JavaScript API Reference.

`AdminInitiateAuth`

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

SDK per JavaScript (v3)

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

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
```

```
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

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

- Per i dettagli sull'API, AdminInitiateAuthconsulta AWS SDK for JavaScript API Reference.

`AdminRespondToAuthChallenge`

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

SDK per JavaScript (v3)

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

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
  username,
  totp,
  session,
}) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AdminRespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: totp,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });
```

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

- Per i dettagli sull'API, AdminRespondToAuthChallengeconsulta AWS SDK for JavaScript API Reference.

`AssociateSoftwareToken`

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

SDK per JavaScript (v3)

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

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

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

- Per i dettagli sull'API, AssociateSoftwareTokenconsulta AWS SDK for JavaScript API Reference.

`ConfirmDevice`

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

SDK per JavaScript (v3)

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

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmDeviceCommand({
    DeviceKey: deviceKey,
    AccessToken: accessToken,
    DeviceSecretVerifierConfig: {
      PasswordVerifier: passwordVerifier,
      Salt: salt,
    },
  });

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

- Per i dettagli sull'API, ConfirmDeviceconsulta AWS SDK for JavaScript API Reference.

`ConfirmSignUp`

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

SDK per JavaScript (v3)

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

```
const confirmSignUp = ({ clientId, username, code }) => {
```

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

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

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

- Per i dettagli sull'API, ConfirmSignUpconsulta AWS SDK for JavaScript API Reference.

`DeleteUser`

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

SDK per JavaScript (v3)

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

```
/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
```

```
    return [null, err];
  }
};
```

- Per i dettagli sull'API, DeleteUserconsulta AWS SDK for JavaScript API Reference.

`InitiateAuth`

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

SDK per JavaScript (v3)

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

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

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
    AuthParameters: {
      USERNAME: username,
      PASSWORD: password,
    },
    ClientId: clientId,
  });

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

- Per i dettagli sull'API, InitiateAuthconsulta AWS SDK for JavaScript API Reference.

`ListUsers`

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

SDK per JavaScript (v3)

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

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

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

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

- Per i dettagli sull'API, ListUsersconsulta AWS SDK for JavaScript API Reference.

`ResendConfirmationCode`

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

SDK per JavaScript (v3)

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

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

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

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

- Per i dettagli sull'API, ResendConfirmationCodeconsulta AWS SDK for JavaScript API Reference.

`RespondToAuthChallenge`

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

SDK per JavaScript (v3)

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

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

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
```

```
};
```

- Per i dettagli sull'API, RespondToAuthChallengeconsulta AWS SDK for JavaScript API Reference.

`SignUp`

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

SDK per JavaScript (v3)

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

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

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

- Per i dettagli sull'API, SignUpconsulta AWS SDK for JavaScript API Reference.

`UpdateUserPool`

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

SDK per JavaScript (v3)

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

```
/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Per i dettagli sull'API, UpdateUserPoolconsulta AWS SDK for JavaScript API Reference.

`VerifySoftwareToken`

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

SDK per JavaScript (v3)

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

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

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

- Per i dettagli sull'API, VerifySoftwareTokenconsulta AWS SDK for JavaScript API Reference.

Scenari

Confermare automaticamente gli utenti noti con una funzione Lambda

L’esempio di codice seguente mostra come confermare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PreSignUp`.
- Registra un utente con Amazon Cognito.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e conferma automaticamente gli utenti noti.
- Accedi come nuovo utente, quindi elimina le risorse.

SDK per JavaScript (v3)

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

Configura un'esecuzione interattiva dello "Scenario". Gli esempi JavaScript (v3) condividono uno Scenario runner per semplificare esempi complessi. Il codice sorgente completo è attivo. GitHub

```
import { AutoConfirm } from "./scenario-auto-confirm.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      UserEmail: "test_email_1@example.com",
    },
    {
      UserName: "test_user_2",
      UserEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      UserEmail: "test_email_3@example.com",
    },
  ],
};
```

```
/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito
 authentication behavior.",
  });
}
```

Questo scenario dimostra la conferma automatica di un utente noto. Orchestra i passaggi di esempio.

```
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
} from "./steps-common.js";
```

```
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
```

```
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (/** @type {State} */ state) => {
    const [_, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
```

```
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
 "${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (/** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (/** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (/** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
```

```
    }

    if (user) {
      state.errors.push(
        new Error(
          `The user "${state.selectedUser}" already exists in the user pool
 "${state.UserPoolId}".`,
        ),
      );
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
 numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (/** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({
        region: state.stackRegion,
        userPoolClientId: state.UserPoolClientId,
        username: state.selectedUser,
        email: state.users.find((u) => u.UserName === state.selectedUser)
          .UserEmail,
        password,
      });

    let [_, err] = await signUp(state.password);
```

```
    while (err?.name === "InvalidPasswordException") {
      console.warn("The password you entered was invalid.");
      await createPassword.handle(state);
      [_, err] = await signUp(state.password);
    }

    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (/** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
    if (logStreamErr) {
      state.errors.push(logStreamErr);
      return;
    }

    console.log(
      `Getting some recent events from log stream "${logStream.logStreamName}"`,
    );
    const [logEvents, logEventsErr] = await getLogEvents({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
      eventCount: 10,
```

```
      logStreamName: logStream.logStreamName,
    });
    if (logEventsErr) {
      state.errors.push(logEventsErr);
      return;
    }

    console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });

    if (err?.name === "PasswordResetRequiredException") {
      state.errors.push(new Error("Please reset your password."));
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
```

```
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with: ${state.token.slice(0,
 11)}`,
  { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [_, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);

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}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);
```

```
export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
  );
```

Questi sono passaggi condivisi con altri scenari.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";
```

```
export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);
```

Un handler per il trigger `PreSignUp` con una funzione Lambda.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "./user-repository";
```

```
import { DynamoDBUserRepository } from "./user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user '${event.userName}'`,
    );

    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }

    const eventEmail = this.getEventUserEmail(event);
    console.log(`Looking up email ${eventEmail}.`);
    const storedUserInfo =
      await this.userRepository.getUserInfoByEmail(eventEmail);

    if (!storedUserInfo) {
      console.log(
        `Email ${eventEmail} not found. Email verification is required.`,
      );
      return event;
    }

    if (storedUserInfo.UserName !== event.userName) {
      console.log(
```

```
        `UserEmail ${eventEmail} found, but stored UserName
 '${storedUserInfo.UserName}' does not match supplied UserName '${event.userName}'.
 Verification is required.`,
      );
    } else {
      console.log(
        `UserEmail ${eventEmail} found with matching UserName
 ${storedUserInfo.UserName}. User is confirmed.`,
      );
      event.response.autoConfirmUser = true;
      event.response.autoVerifyEmail = true;
    }
    return event;
  }
}

const createPreSignUpHandler = (): PreSignUpHandler => {
  const tableName = process.env.TABLE_NAME;
  if (!tableName) {
    throw new Error("TABLE_NAME environment variable is not set");
  }

  const userRepository = new DynamoDBUserRepository(tableName);
  return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

Modulo di azioni CloudWatch Logs.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
```

```
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
 unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[] |
 null, unknown]>}
 */
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
```

```
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};
```

Modulo di azioni Amazon Cognito.

```
import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
```

```
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 *   password: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const response = await cognitoClient.send(
      new SignUpCommand({
        ClientId: userPoolClientId,
```

```
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 flow.
 * @param {{ region: string, clientId: string, username: string, password: string }}
 config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
```

```
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

Modulo di azioni DynamoDB.

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

/**
 * Populate a DynamoDB table with provide items.
```

```
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }}
 config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
 null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Registrazione di un utente a un pool di utenti che richiede l'autenticazione MFA

L'esempio di codice seguente mostra come:

- Registra e conferma un utente con nome utente, password e indirizzo e-mail.
- Configura l'autenticazione a più fattori associando un'applicazione MFA all'utente.

- Accedi utilizzando una password e un codice MFA.

SDK per JavaScript (v3)

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

Per un'esperienza ottimale, clona il GitHub repository ed esegui questo esempio. Il codice seguente rappresenta un esempio dell'applicazione completa.

```
import { logger } 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);
    logger.log("Signing up.");
    await signUp({ clientId, username, password, email });
    logger.log(`Signed up. A confirmation email has been sent to: ${email}.`);
    logger.log(
      `Run 'confirm-sign-up ${username} <code>' to confirm your account.`,
    );
  } catch (err) {
    logger.error(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 { logger } 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);
    logger.log("Confirming user.");
    await confirmSignUp({ clientId, username, code });
    logger.log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
    );
  } catch (err) {
    logger.error(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 { logger } 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);

    logger.log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      logger.log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }
```

```
    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      logger.log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    logger.error(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 { logger } 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);

    logger.log("Successfully authenticated.");
  } catch (err) {
    logger.error(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 { logger } 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);

    logger.log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    logger.log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    logger.error(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});
```

```
  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - 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.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

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.

SDK per (v3 JavaScript)

> Mostra come utilizzare Amazon Transcribe per creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia i risultati per e-mail tramite Amazon Simple Email Service (Amazon SES).
>
> Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub
>
> Servizi utilizzati in questo esempio
>
> - Amazon Comprehend
> - Amazon SES
> - Amazon Transcribe
> - Amazon Translate

Creazione di un chatbot Amazon Lex

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

SDK per JavaScript (v3)

Mostra come utilizzare l'API Amazon Lex per creare un chatbot all'interno di un'applicazione web per coinvolgere i visitatori del sito web.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo Costruire un chatbot Amazon Lex nella guida per gli AWS SDK for JavaScript sviluppatori.

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

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.

SDK per JavaScript (v3)

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in. GitHub I seguenti estratti mostrano come AWS SDK for JavaScript viene utilizzato all'interno delle funzioni Lambda.

```
import {
```

```
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";
```

```
/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
```

```
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});
```

```
  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

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 link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Esempi serverless

Esempi serverless

Invocare una funzione Lambda da un trigger Amazon DocumentDB

L'esempio di codice seguente 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.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. 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 con SDK JavaScript for (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con DynamoDB.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni
- Scenari
- Esempi serverless

Nozioni di base

Hello DynamoDB

L'esempio di codice seguente mostra come iniziare a utilizzare DynamoDB.

SDK per (v3) JavaScript

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

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 dettagli sulle API, consulta ListTablesla sezione API Reference.AWS SDK for JavaScript

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea una tabella in grado di contenere i dati del filmato.
- Inserire, ottenere e aggiornare un singolo filmato nella tabella.

- Scrivere i dati del filmato nella tabella da un file JSON di esempio.
- Eseguire una query sui filmati che sono stati rilasciati in un dato anno.
- Cerca i filmati che sono stati distribuiti in diversi anni.
- Elimina un filmato dalla tabella, quindi elimina la tabella.

SDK per JavaScript (v3)

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

```
import { readFileSync } from "node: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 docClient.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
}
log("Movies added.");

/**
 * Query for movies by year.
 */

log("Querying for all movies from 1981.");
const paginatedQuery = paginateQuery(
  { client: docClient },
```

```
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression. Scan
 will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
```

```
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Azioni

`BatchExecuteStatement`

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

SDK per JavaScript (v3)

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

Creazione di un batch di elementi mediante PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });

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

Ottenimento di un batch di elementi mediante PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

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

Aggiornamento di un batch di elementi mediante PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

```
export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

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

Eliminazione di un batch di elementi mediante PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
```

```
  });

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

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK for JavaScript API Reference.

`BatchGetItem`

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

SDK per JavaScript (v3)

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

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta BatchGet.

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

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

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
```

```
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sulle API, consulta la BatchGetItemsezione AWS SDK for JavaScript API Reference.

`BatchWriteItem`

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

SDK per JavaScript (v3)

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

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta BatchWrite.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node:fs";
```

```
// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

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

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year' is
 recommended
        // to account for duplicate titles.
        BatchWriteMoviesTable: putRequests,
      },
    });

    await docClient.send(command);
  }
};
```

- Per i dettagli sulle API, consulta la BatchWriteItemsezione AWS SDK for JavaScript API Reference.

`CreateTable`

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

SDK per JavaScript (v3)

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

```
import { 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",
      },
    ],
```

```
    BillingMode: "PAY_PER_REQUEST",
  });

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateTableconsulta AWS SDK for JavaScript API Reference.

`DeleteItem`

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

SDK per JavaScript (v3)

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

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta DeleteCommand.

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

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

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

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sulle API, consulta la DeleteItemsezione AWS SDK for JavaScript API Reference.

`DeleteTable`

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

SDK per JavaScript (v3)

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

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

const client = new DynamoDBClient({});

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

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

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK for JavaScript API Reference.

`DescribeTable`

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

SDK per JavaScript (v3)

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

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

const client = new DynamoDBClient({});

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

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DescribeTableconsulta AWS SDK for JavaScript API Reference.

`DescribeTimeToLive`

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

SDK per JavaScript (v3)

Per descrivere la configurazione TTL in una tabella DynamoDB esistente con AWS SDK for JavaScript.

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

export const describeTTL = 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;
    }
}

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DescribeTimeToLiveReference.

`ExecuteStatement`

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

SDK per JavaScript (v3)

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

Creazione di un elemento mediante PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

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

Ottenimento di un elemento mediante PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

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

```
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

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

Aggiornamento di un elemento mediante PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

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

Eliminazione di un elemento mediante PartiQL.

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

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

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

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

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

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK for JavaScript API Reference.

`GetItem`

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

SDK per JavaScript (v3)

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

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta GetCommand.

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

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

```
const docClient = DynamoDBDocumentClient.from(client);

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

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

- Per i dettagli sulle API, consulta la GetItemsezione AWS SDK for JavaScript API Reference.

`ListTables`

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

SDK per JavaScript (v3)

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

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

const client = new DynamoDBClient({});

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

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListTablesconsulta AWS SDK for JavaScript API Reference.

`PutItem`

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

SDK per JavaScript (v3)

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

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta PutCommand.

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

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

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

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

- Per i dettagli sulle API, consulta la PutItemsezione AWS SDK for JavaScript API Reference.

`Query`

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

SDK per JavaScript (v3)

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

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta QueryCommand.

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

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

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

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

`Scan`

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

SDK per JavaScript (v3)

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

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta ScanCommand.

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

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

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- Per informazioni dettagliate sull'API, consulta Scan nella documentazione di riferimento dell'API AWS SDK for JavaScript .

`UpdateItem`

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

SDK per JavaScript (v3)

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

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta UpdateCommand.

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

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

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

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

- Per i dettagli sulle API, consulta la UpdateItemsezione AWS SDK for JavaScript API Reference.

`UpdateTimeToLive`

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

SDK per JavaScript (v3)

Abilitare TTL in una tabella DynamoDB esistente.

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

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') => {

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

    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;
    }
};

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Disabilitare TTL in una tabella DynamoDB esistente.

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

export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1') => {

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

    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;
    }
};

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

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

Scenari

Costruisci un'app per inviare dati a una tabella DynamoDB

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

SDK per JavaScript (v3)

Questo esempio mostra come creare un'app che consenta agli utenti di inviare dati a una tabella Amazon DynamoDB e un messaggio di testo all'amministratore utilizzando Amazon Simple Notification Service (Amazon SNS).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SNS

Confrontare più valori con un unico attributo

L'esempio di codice seguente mostra come confrontare più valori con un unico attributo in DynamoDB.

- Utilizzare l'operatore IN per confrontare più valori con un unico attributo.
- Confrontare l'operatore IN con più condizioni OR.
- Comprendere i vantaggi in termini di prestazioni e complessità di espressione derivanti dall'utilizzo dell'operatore IN.

SDK per JavaScript (v3)

Confronta più valori con un singolo attributo utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
```

```
  ScanCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Query or scan a DynamoDB table to find items where an attribute matches any value
 from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 attribute
 * against multiple possible values, which is more efficient than using multiple OR
 conditions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against the
 values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key attribute
 for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the matching
 items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the filter expression using the IN operator
  const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
 `:val${index}`).join(', ')})`;

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
```

```
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
```

```
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against the
 values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key attribute
 for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the matching
 items
 */
async function compareWithOrConditions(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }

  // Create the filter expression using multiple OR conditions
  const filterConditions = valuesList.map((_, index) => `${attributeName} = :val
${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
```

```
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;
```

```
    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}
```

Esempio di utilizzo del confronto di più valori con AWS SDK for JavaScript.

```
/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const attributeName = "Category";
```

```
 const valuesList = ["Electronics", "Computers", "Accessories"];

 console.log(`Searching for products in any of these categories:
${valuesList.join(', ')}`);

 try {
   // Using the IN operator (recommended approach)
   console.log("\nApproach 1: Using the IN operator");
   const response = await compareMultipleValues(
     config,
     tableName,
     attributeName,
     valuesList
   );

   console.log(`Found ${response.Count} products in the specified categories`);

   // Using multiple OR conditions (alternative approach)
   console.log("\nApproach 2: Using multiple OR conditions");
   const response2 = await compareWithOrConditions(
     config,
     tableName,
     attributeName,
     valuesList
   );

   console.log(`Found ${response2.Count} products in the specified categories`);

   // Example with a query operation
   console.log("\nQuerying a specific manufacturer's products in multiple
categories");
   const partitionKeyName = "Manufacturer";
   const partitionKeyValue = "Acme";

   const response3 = await compareMultipleValues(
     config,
     tableName,
     attributeName,
     valuesList,
     partitionKeyName,
     partitionKeyValue
   );
```

```
    console.log(`Found ${response3.Count} Acme products in the specified
 categories`);

    // Explain the benefits of using the IN operator
    console.log("\nBenefits of using the IN operator:");
    console.log("1. More concise expression compared to multiple OR conditions");
    console.log("2. Better readability and maintainability");
    console.log("3. Potentially better performance with large value lists");
    console.log("4. Simpler code that's less prone to errors");
    console.log("5. Easier to modify when adding or removing values");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - Query
  - Scan

Eseguire l'aggiornamento condizionale del TTL di un elemento

L'esempio di codice seguente mostra come aggiornare in modo condizionale il valore TTL di un elemento.

SDK per JavaScript (v3)

Aggiorna il TTL di 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";

export const updateItemConditional = async (tableName, partitionKey, sortKey, region
 = 'us-east-1', newAttribute = 'default-value') => {
    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;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-sort-
key-value');
```

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

Contare gli operatori di espressione

L’esempio di codice seguente mostra come conteggiare gli operatori delle espressioni in DynamoDB.

- Comprende il limite dell’operatore 300 di DynamoDB.

- Contare gli operatori nelle espressioni complesse.
- Ottimizzare le espressioni affinché rientrino nei limiti.

SDK per JavaScript (v3)

Per descrivere il conteggio degli operatori di espressione con AWS SDK for JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 conditions
 * @returns {Object} - Object containing the filter expression and attribute values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};

  // Generate the specified number of conditions
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
```

```
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        condition = `attribute${i} < ${valueKey}`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
      case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
      case 4:
        condition = `attribute_exists(attribute${i})`;
        break;
    }

    conditions.push(condition);
  }

  // Join the conditions with AND or OR
  const operator = useAnd ? " AND " : " OR ";
  const filterExpression = conditions.join(operator);

  // Calculate the operator count
  // Each condition has 1 operator (=, >, <, contains, attribute_exists)
  // Each AND or OR between conditions is 1 operator
  const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount -
 1 : 0);

  return {
    filterExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute values
```

```
 */
function createComplexUpdateExpression(operationsCount) {
  // Initialize the expression parts and attribute values
  const setOperations = [];
  const expressionAttributeValues = {};

  // Generate the specified number of SET operations
  for (let i = 0; i < operationsCount; i++) {
    // Alternate between different types of SET operations
    let operation;
    const valueKey = `:val${i}`;

    switch (i % 3) {
      case 0:
        // Simple assignment (1 operator: =)
        operation = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        // Addition (2 operators: = and +)
        operation = `attribute${i} = attribute${i} + ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        // Conditional assignment with if_not_exists (2 operators: = and
 if_not_exists)
        operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
    }

    setOperations.push(operation);
  }

  // Create the update expression
  const updateExpression = `SET ${setOperations.join(", ")}`;

  // Calculate the operator count
  // Each operation has 1-2 operators as noted above
  let operatorCount = 0;
  for (let i = 0; i < operationsCount; i++) {
    operatorCount += (i % 3 === 0) ? 1 : 2;
  }
```

```
  return {
    updateExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
async function testOperatorLimit(
  config,
  tableName,
  key,
  operatorCount
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create a complex update expression with the specified operator count
  const { updateExpression, expressionAttributeValues, operatorCount: actualCount }
 =
    createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to get
 close to target count

  console.log(`Generated update expression with approximately ${actualCount}
 operators`);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
```

```
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Attempt the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      message: `Operation succeeded with ${actualCount} operators`,
      data: response
    };
  } catch (error) {
    // Check if the error is due to exceeding the operator limit
    if (error.name === "ValidationException" &&
        error.message.includes("too many operators")) {
      return {
        success: false,
        message: `Operation failed: ${error.message}`,
        operatorCount: actualCount
      };
    }

    // Return other errors
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      error
    };
  }
}

/**
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
```

```
  config,
  tableName,
  key,
  totalOperations
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Calculate how many operations we can safely include in each batch
  // Using 150 as a conservative limit (well below 300)
  const operationsPerBatch = 100;
  const batchCount = Math.ceil(totalOperations / operationsPerBatch);

  console.log(`Breaking down ${totalOperations} operations into ${batchCount}
batches`);

  const results = [];

  // Process each batch
  for (let batch = 0; batch < batchCount; batch++) {
    // Calculate the operations for this batch
    const batchStart = batch * operationsPerBatch;
    const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
    const batchSize = batchEnd - batchStart;

    console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
operations`);

    // Create an update expression for this batch
    const { updateExpression, expressionAttributeValues, operatorCount } =
      createComplexUpdateExpression(batchSize);

    // Define the update parameters
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: updateExpression,
      ExpressionAttributeValues: expressionAttributeValues,
      ReturnValues: "UPDATED_NEW"
    };

    try {
      // Perform the update operation for this batch
```

```
      const response = await docClient.send(new UpdateCommand(params));

      results.push({
        batch: batch + 1,
        success: true,
        operatorCount,
        attributes: response.Attributes
      });
    } catch (error) {
      results.push({
        batch: batch + 1,
        success: false,
        operatorCount,
        error: error.message
      });

      // Stop processing if an error occurs
      break;
    }
  }

  return {
    totalBatches: batchCount,
    results
  };
}

/**
 * Count operators in a DynamoDB expression based on the rules in the documentation.
 *
 * This function demonstrates how operators are counted according to the
 * DynamoDB documentation.
 *
 * @param {string} expression - The DynamoDB expression to analyze
 * @returns {Object} - Breakdown of operator counts
 */
function countOperatorsInExpression(expression) {
  // Initialize counters for different operator types
  const counts = {
    comparisonOperators: 0,
    logicalOperators: 0,
    functions: 0,
    arithmeticOperators: 0,
    specialOperators: 0,
```

```
    total: 0
  };

  // Count comparison operators (=, <>, <, <=, >, >=)
  const comparisonRegex = /[^<>]=[^=]|<>|<=|>=|[^<]>[^=]|[^>]<[^=]/g;
  const comparisonMatches = expression.match(comparisonRegex) || [];
  counts.comparisonOperators = comparisonMatches.length;

  // Count logical operators (AND, OR, NOT)
  const andMatches = expression.match(/\bAND\b/g) || [];
  const orMatches = expression.match(/\bOR\b/g) || [];
  const notMatches = expression.match(/\bNOT\b/g) || [];
  counts.logicalOperators = andMatches.length + orMatches.length +
 notMatches.length;

  // Count functions (attribute_exists, attribute_not_exists, attribute_type,
 begins_with, contains, size)
  const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
  const functionMatches = expression.match(functionRegex) || [];
  counts.functions = functionMatches.length;

  // Count arithmetic operators (+ and -)
  const arithmeticMatches = expression.match(/[a-zA-Z0-9_)\]]\s*[\+\-]\s*[a-zA-
Z0-9_(:]/g) || [];
  counts.arithmeticOperators = arithmeticMatches.length;

  // Count special operators (BETWEEN, IN)
  const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
  const inMatches = expression.match(/\bIN\b/g) || [];
  counts.specialOperators = betweenMatches.length + inMatches.length;

  // Add extra operators for BETWEEN (each BETWEEN includes an AND)
  counts.logicalOperators += betweenMatches.length;

  // Calculate total
  counts.total = counts.comparisonOperators +
                 counts.logicalOperators +
                 counts.functions +
                 counts.arithmeticOperators +
                 counts.specialOperators;

  return counts;
}
```

Esempio di utilizzo dell'operatore di espressione Counting with. AWS SDK for JavaScript

```
/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating DynamoDB expression operator counting and the 300
 operator limit");

  try {
    // Example 1: Analyze a simple expression
    console.log("\nExample 1: Analyzing a simple expression");
    const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
 (:cat1, :cat2, :cat3)";
    const simpleCount = countOperatorsInExpression(simpleExpression);

    console.log(`Expression: ${simpleExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
    console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
    console.log(`- Functions: ${simpleCount.functions}`);
    console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
    console.log(`- Special operators: ${simpleCount.specialOperators}`);
    console.log(`- Total operators: ${simpleCount.total}`);

    // Example 2: Analyze a complex expression
    console.log("\nExample 2: Analyzing a complex expression");
    const complexExpression =
      "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
      "(Price > :price AND contains(Description, :keyword) AND " +
      "(Rating >= :minRating OR Reviews > :minReviews))";
    const complexCount = countOperatorsInExpression(complexExpression);

    console.log(`Expression: ${complexExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
```

```
    console.log(`- Logical operators: ${complexCount.logicalOperators}`);
    console.log(`- Functions: ${complexCount.functions}`);
    console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
    console.log(`- Special operators: ${complexCount.specialOperators}`);
    console.log(`- Total operators: ${complexCount.total}`);

    // Example 3: Test approaching the operator limit
    console.log("\nExample 3: Testing an expression approaching the operator
limit");
    const approachingLimit = await testOperatorLimit(config, tableName, key, 290);
    console.log(approachingLimit.message);

    // Example 4: Test exceeding the operator limit
    console.log("\nExample 4: Testing an expression exceeding the operator limit");
    const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
    console.log(exceedingLimit.message);

    // Example 5: Breaking down a complex expression
    console.log("\nExample 5: Breaking down a complex expression into multiple
operations");
    const breakdownResult = await breakDownComplexExpression(config, tableName, key,
500);
    console.log(`Processed ${breakdownResult.results.length} of
${breakdownResult.totalBatches} batches`);

    // Explain the operator counting rules
    console.log("\nKey points about DynamoDB expression operator counting:");
    console.log("1. The maximum number of operators in any expression is 300");
    console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
operator");
    console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
    console.log("4. Each function call (attribute_exists, contains, etc.) counts as
1 operator");
    console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
    console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND within
it)");
    console.log("7. IN counts as 1 operator regardless of the number of values");
    console.log("8. Parentheses for grouping and attribute paths don't count as
operators");
    console.log("9. When you exceed the limit, the error always reports '301
operators'");
    console.log("10. For complex operations, break them into multiple smaller
operations");
```

```
  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Per i dettagli sull'API, consulta la sezione UpdateItem AWS SDK for JavaScriptAPI Reference.

Creazione di un'applicazione serverless per gestire foto

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

SDK per JavaScript (v3)

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

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

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Creare una tabella con throughput a caldo abilitato

L'esempio di codice seguente mostra come creare una tabella con throughput a caldo abilitato.

SDK per JavaScript (v3)

Creare una tabella DynamoDB con throughput a caldo con AWS SDK for JavaScript.

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

```
export async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
```

```
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
    const response = await ddbClient.send(command);
    console.log(response);
    return response;
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
  'example-table',
  'pk',
  'sk',
  'gsiKey',
  'data',
  10, 10, 5, 5,
  'example-index',
  5, 5, 2, 2
);
*/
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API CreateTableReference.

Creare un elemento con un TTL

L'esempio di codice seguente mostra come creare un elemento con un valore TTL.

SDK per JavaScript (v3)

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

export 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, err);
            throw err;
        } else {
```

```
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API PutItemReference.

Elimina i dati utilizzando istruzioni PartiQL DELETE

L'esempio di codice seguente mostra come eliminare i dati mediante le istruzioni PartiQL DELETE.

SDK per JavaScript (v3)

Eliminare elementi da una tabella DynamoDB utilizzando le istruzioni PartiQL DELETE con. AWS SDK for JavaScript

```
/**
 * This example demonstrates how to delete items from a DynamoDB table using
 PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByPartitionKey = async (
```

```
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
```

```
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item with a condition to ensure the delete only happens if a condition
 is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${conditionAttribute} = ?`,
    Parameters: [partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
```

```
    console.log("Item deleted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item with condition:", err);
    throw err;
  }
};

/**
 * Batch delete multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param keys - Array of objects containing key information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchDeleteItems = async (
  tableName: string,
  keys: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    sortKeyName?: string;
    sortKeyValue?: string | number;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each delete
  const statements = keys.map((key) => {
    if (key.sortKeyName && key.sortKeyValue !== undefined) {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ? AND
 ${key.sortKeyName} = ?`,
        Parameters: [key.partitionKeyValue, key.sortKeyValue],
      };
    } else {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?`,
        Parameters: [key.partitionKeyValue],
      };
    }
  });

  const params = {
```

```
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch deleted successfully");
    return data;
  } catch (err) {
    console.error("Error batch deleting items:", err);
    throw err;
  }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemsByFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items deleted by filter successfully");
    return data;
  } catch (err) {
    console.error("Error deleting items by filter:", err);
    throw err;
  }
};
```

```
/**
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {
  // Delete an item by partition key (simple primary key)
  await deleteItemByPartitionKey("UsersTable", "userId", "user123");

  // Delete an item by composite key (partition key + sort key)
  await deleteItemByCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789"
  );

  // Delete with a condition
  await deleteItemWithCondition(
    "UsersTable",
    "userId",
    "user789",
    "userStatus",
    "inactive"
  );

  // Batch delete multiple items
  await batchDeleteItems("UsersTable", [
    { partitionKeyName: "userId", partitionKeyValue: "user234" },
    { partitionKeyName: "userId", partitionKeyValue: "user345" },
  ]);

  // Batch delete items with composite keys
  await batchDeleteItems("OrdersTable", [
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order567",
      sortKeyName: "productId",
      sortKeyValue: "prod123",
    },
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order678",
      sortKeyName: "productId",
```

```
      sortKeyValue: "prod456",
    },
  ]);

  // Delete items by filter (use with caution)
  await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Inserire dati con istruzioni PartiQL INSERT

L'esempio di codice seguente mostra come inserire i dati mediante le istruzioni PartiQL INSERT.

SDK per (v3) JavaScript

Inserisci elementi in una tabella DynamoDB utilizzando le istruzioni PartiQL INSERT con. AWS SDK for JavaScript

```
/**
 * This example demonstrates how to insert items into a DynamoDB table using
 PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
```

```
 */
export const insertItem = async (tableName: string, item: Record<string, any>) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Convert the item to a string representation for PartiQL
  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item:", err);
    throw err;
  }
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string,
 any>[]) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
    return {
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
```

```
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
    return data;
  } catch (err) {
    console.error("Error batch inserting items:", err);
    throw err;
  }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
  tableName: string,
  item: Record<string, any>,
  partitionKeyName: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
  const partitionKeyValue = JSON.stringify(item[partitionKeyName]);

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
 attribute_not_exists(${partitionKeyName})`,
    Parameters: [{ S: partitionKeyValue }],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted with condition successfully");
    return data;
  } catch (err) {
```

```
    console.error("Error inserting item with condition:", err);
    throw err;
  }
};

/**
 * Example usage showing how to insert items with different index types
 */
export const insertExamples = async () => {
  // Example table with a simple primary key (just partition key)
  const simpleKeyItem = {
    userId: "user123",
    name: "John Doe",
    email: "john@example.com",
  };
  await insertItem("UsersTable", simpleKeyItem);

  // Example table with composite key (partition key + sort key)
  const compositeKeyItem = {
    orderId: "order456",
    productId: "prod789",
    quantity: 2,
    price: 29.99,
  };
  await insertItem("OrdersTable", compositeKeyItem);

  // Example with Global Secondary Index (GSI)
  // The GSI might be on the email attribute
  const gsiItem = {
    userId: "user789",
    email: "jane@example.com",
    name: "Jane Smith",
    userType: "premium",  // This could be part of a GSI
  };
  await insertItem("UsersTable", gsiItem);

  // Example with Local Secondary Index (LSI)
  // LSI uses the same partition key but different sort key
  const lsiItem = {
    orderId: "order567",  // Partition key
    productId: "prod123", // Sort key for the table
    orderDate: "2023-11-15", // Potential sort key for an LSI
    quantity: 1,
    price: 19.99,
```

```
  };
  await insertItem("OrdersTable", lsiItem);

  // Batch insert example with multiple items
  const batchItems = [
    {
      userId: "user234",
      name: "Alice Johnson",
      email: "alice@example.com",
    },
    {
      userId: "user345",
      name: "Bob Williams",
      email: "bob@example.com",
    },
  ];
  await batchInsertItems("UsersTable", batchItems);
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Invocazione di una funzione Lambda da un browser

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

SDK per JavaScript (v3)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente. Questa app utilizza la versione 3. AWS SDK for JavaScript

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

Eseguire operazioni di query avanzate

L'esempio di codice seguente mostra come eseguire operazioni di query avanzate in DynamoDB.

- Eseguire query si tabelle mediante varie tecniche di filtro e condizione.
- Implementare l'impaginazione per set di risultati di grandi dimensioni.
- Utilizzare gli indici secondari globali per modelli di accesso alternativi.
- Applicare controlli di consistenza in base ai requisiti dell'applicazione.

SDK per JavaScript (v3)

Interrogazione con letture fortemente coerenti utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
```

```
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

Esegui una query utilizzando un indice secondario globale con AWS SDK for JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
```

```
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
```

```
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

Interrogazione con impaginazione utilizzando AWS SDK for JavaScript.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);
```

```
    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages
```

```
    console.log(`Query complete. Retrieved ${allItems.length} items in ${pageCount}
 pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

Interrogazione con filtri complessi utilizzando AWS SDK for JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
```

```
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
```

```
}
```

Interrogazione con un'espressione di filtro costruita dinamicamente utilizzando AWS SDK for JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
```

```
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Esecuzione di operazioni di tipo elenco

L'esempio di codice seguente mostra come eseguire operazioni elenco in DynamoDB.

- Aggiungere elementi a un attributo di elenco.
- Rimuovere elementi da un attributo di elenco.

- Aggiornare elementi specifici in un elenco in base all'indice.
- Utilizzare le funzioni di aggiunta all'elenco e indicizzazione di elenchi.

SDK per JavaScript (v3)

Dimostra le operazioni sugli elenchi utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `SET ${listName} =
 list_append(if_not_exists(${listName}, :empty_list), :values)`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `SET ${listName} = list_append(:values,
 if_not_exists(${listName}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using index notation
  const params = {
```

```
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${index}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
  config,
  tableName,
  key,
  listName,
  index
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using REMOVE with index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `REMOVE ${listName}[${index}]`,
```

```
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Concatenate two lists.
 *
 * This function demonstrates how to concatenate two lists using the list_append
 function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${resultListName} =
 list_append(if_not_exists(${listName1}, :empty_list),
 if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
```

```
      ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a nested list structure.
 *
 * This function demonstrates how to create and work with nested lists.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} nestedLists - An array of arrays to create a nested list structure
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createNestedList(
  config,
  tableName,
  key,
  listName,
  nestedLists
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters to create a nested list
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = :nested_lists`,
    ExpressionAttributeValues: {
      ":nested_lists": nestedLists
    },
    ReturnValues: "UPDATED_NEW"
  };
```

```
  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an element in a nested list.
 *
 * This function demonstrates how to update an element in a nested list
 * using multiple index notations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} outerIndex - The index in the outer list
 * @param {number} innerIndex - The index in the inner list
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedListElement(
  config,
  tableName,
  key,
  listName,
  outerIndex,
  innerIndex,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using multiple index notations
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
```

```
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Esempio di utilizzo delle operazioni di elenco con AWS SDK for JavaScript.

```
/**
 * Example of how to work with lists in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

  console.log("Demonstrating list operations in DynamoDB");

  try {
    // Example 1: Append elements to a list
    console.log("\nExample 1: Appending elements to a list");
    const response1 = await appendToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["laptop", "headphones", "monitor"]
    );

    console.log("Appended to list:", response1.Attributes);

    // Example 2: Prepend elements to a list
    console.log("\nExample 2: Prepending elements to a list");
    const response2 = await prependToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["keyboard", "mouse"]
    );

    console.log("Prepended to list:", response2.Attributes);

    // Get the current state of the item
    let currentItem = await getItem(config, tableName, key);
    console.log("\nCurrent state of RecentSearches:", currentItem?.RecentSearches);

    // Example 3: Update a specific element in a list
    console.log("\nExample 3: Updating a specific element in a list");
    const response3 = await updateListElement(
```

```
    config,
    tableName,
    key,
    "RecentSearches",
    0, // Update the first element
    "mechanical keyboard" // New value
  );

  console.log("Updated list element:", response3.Attributes);

  // Example 4: Remove an element from a list
  console.log("\nExample 4: Removing an element from a list");
  const response4 = await removeListElement(
    config,
    tableName,
    key,
    "RecentSearches",
    2 // Remove the third element
  );

  console.log("List after removing element:", response4.Attributes);

  // Example 5: Create and concatenate lists
  console.log("\nExample 5: Creating and concatenating lists");

  // First, create two separate lists
  await updateWithMultipleActions(
    config,
    tableName,
    key,
    "SET WishList = :wishlist, SavedItems = :saveditems",
    null,
    {
      ":wishlist": ["gaming laptop", "wireless earbuds"],
      ":saveditems": ["smartphone", "tablet"]
    }
  );

  // Then, concatenate them
  const response5 = await concatenateLists(
    config,
    tableName,
    key,
    "WishList",
```

```
    "SavedItems",
    "AllItems"
  );

  console.log("Concatenated lists:", response5.Attributes);

  // Example 6: Create a nested list structure
  console.log("\nExample 6: Creating a nested list structure");
  const response6 = await createNestedList(
    config,
    tableName,
    key,
    "Categories",
    [
      ["Electronics", "Computers", "Accessories"],
      ["Books", "Magazines", "E-books"],
      ["Clothing", "Shoes", "Watches"]
    ]
  );

  console.log("Created nested list:", response6.Attributes);

  // Example 7: Update an element in a nested list
  console.log("\nExample 7: Updating an element in a nested list");
  const response7 = await updateNestedListElement(
    config,
    tableName,
    key,
    "Categories",
    0, // First inner list
    1, // Second element in that list
    "Laptops" // New value
  );

  console.log("Updated nested list element:", response7.Attributes);

  // Get the final state of the item
  currentItem = await getItem(config, tableName, key);
  console.log("\nFinal state of the item:", JSON.stringify(currentItem, null, 2));

  // Explain list operations
  console.log("\nKey points about list operations in DynamoDB:");
  console.log("1. Use list_append to add elements to a list");
```

```
    console.log("2. To append elements, use list_append(existingList,
 newElements)");
    console.log("3. To prepend elements, use list_append(newElements,
 existingList)");
    console.log("4. Use if_not_exists to handle cases where the list might not exist
 yet");
    console.log("5. Use index notation (list[0]) to access or update specific
 elements");
    console.log("6. Use REMOVE with index notation to remove elements from a list");
    console.log("7. Lists can contain elements of different types");
    console.log("8. Lists can be nested (lists of lists)");
    console.log("9. Use multiple index notations (list[0][1]) to access nested list
 elements");

  } catch (error) {
    console.error("Error:", error);
  }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
```

```
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK for JavaScript API Reference.

Eseguire operazioni di mappatura

L'esempio di codice seguente mostra come eseguire operazioni di mappatura in DynamoDB.

- Aggiungere e aggiornare gli attributi annidati nelle strutture di mappatura.
- Rimuovere campi specifici dalle mappe.
- Utilizzare attributi di mappatura annidati in profondità.

SDK per JavaScript (v3)

Dimostra le operazioni sulla mappa utilizzando. AWS SDK for JavaScript

```
/**
 * Example of updating map attributes in DynamoDB.
```

```
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update scenarios.
 */

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeSafe(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
```

```
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {
    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update expression
 is invalid")) {

      // Create the map with the specified key-value pair
      const createParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${mapName} = :map`,
        ExpressionAttributeValues: {
          ":map": { [mapKey]: value }
        },
        ReturnValues: "UPDATED_NEW"
      };

      return await docClient.send(new UpdateCommand(createParams));
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
```

```
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeWithIfNotExists(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
 ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":emptyMap": {},
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
```

```
 * @param {string[]} path - The path to the nested attribute as an array of keys
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
      updateExpression += ` ${parentPath} = if_not_exists(${parentPath}, :emptyMap
${i}),`;
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[":value"] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };
```

```
  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 *
 * This function demonstrates how to update multiple fields in a map
 * in a single DynamoDB operation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {Object} updates - Object containing key-value pairs to update
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
  updates
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Define the update parameters
```

```
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
```

```
}

/**
 * Example of how to use the map attribute update functions.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating different approaches to update map attributes in
 DynamoDB");

  try {
    // Example 1: Update a map attribute that might not exist (two-step approach)
    console.log("\nExample 1: Updating a map attribute that might not exist (two-
step approach)");
    const response1 = await updateMapAttributeSafe(
      config,
      tableName,
      key,
      "Preferences",
      "Theme",
      "Dark"
    );

    console.log("Updated preferences:", response1.Attributes);

    // Example 2: Update a map attribute using if_not_exists (elegant approach)
    console.log("\nExample 2: Updating a map attribute using if_not_exists (elegant
 approach)");
    const response2 = await updateMapAttributeWithIfNotExists(
      config,
      tableName,
      key,
      "Settings",
      "NotificationsEnabled",
      true
    );

    console.log("Updated settings:", response2.Attributes);

    // Example 3: Update a deeply nested attribute
```

```
    console.log("\nExample 3: Updating a deeply nested attribute");
    const response3 = await addToNestedMap(
      config,
      tableName,
      key,
      ["Profile", "Address", "City"],
      "Seattle"
    );

    console.log("Updated nested attribute:", response3.Attributes);

    // Example 4: Update multiple fields in a map
    console.log("\nExample 4: Updating multiple fields in a map");
    const response4 = await updateMultipleMapFields(
      config,
      tableName,
      key,
      "ContactInfo",
      {
        Email: "user@example.com",
        Phone: "555-123-4567",
        PreferredContact: "Email"
      }
    );

    console.log("Updated multiple fields:", response4.Attributes);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Explain the benefits of different approaches
    console.log("\nKey points about updating map attributes:");
    console.log("1. Use if_not_exists to handle maps that might not exist");
    console.log("2. Multiple updates can be combined in a single operation");
    console.log("3. Deeply nested attributes require creating parent maps");
    console.log("4. DynamoDB expressions are atomic - the entire update succeeds or
fails");
    console.log("5. Using a single operation is more efficient than multiple
separate updates");

  } catch (error) {
    console.error("Error:", error);
```

```
  }
}

// Export the functions
module.exports = {
  updateMapAttributeSafe,
  updateMapAttributeWithIfNotExists,
  addToNestedMap,
  updateMultipleMapFields,
  getItem,
  exampleUsage
};

// Run the example if this file is executed directly
if (require.main === module) {
  exampleUsage();
}
```

- Per i dettagli sull'API, consulta la UpdateItemsezione AWS SDK for JavaScript API Reference.

Eseguire operazioni su set

L'esempio di codice seguente mostra come eseguire operazioni di impostazione in DynamoDB.

- Aggiungere elementi a un attributo di set.
- Rimuovere elementi da un attributo di set.
- Utilizzare le operazioni ADD e DELETE con i set.

SDK per JavaScript (v3)

Dimostra le operazioni sul set utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
```

```
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
```

```
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove elements from a set attribute.
 *
 * This function demonstrates using the DELETE operation to remove elements from a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to remove from the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
```

```
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using DELETE
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `DELETE ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using SET
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${setName} = :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The new values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function replaceSet(
```

```
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}

/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
    return {
      success: false,
      message: "Set doesn't exist or is already empty",
      item: currentItem
    };
  }
```

```
// Get the set values
const setValues = Array.from(currentItem[setName]);

// If there's only one element left, remove the attribute entirely
if (setValues.length === 1) {
  // Define the update parameters to remove the attribute
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `REMOVE ${setName}`,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  await docClient.send(new UpdateCommand(params));

  return {
    success: true,
    message: "Last element removed, attribute has been deleted",
    removedValue: setValues[0]
  };
} else {
  // Otherwise, remove just the last element
  // Create a set with just the last element
  const lastElement = setValues[setValues.length - 1];
  const setType = typeof lastElement === 'number' ? 'number' : 'string';

  // Remove the last element
  const response = await removeFromSet(
    config,
    tableName,
    key,
    setName,
    [lastElement],
    setType
  );

  return {
    success: true,
    message: "Last element removed, set still contains elements",
    removedValue: lastElement,
    remainingSet: response.Attributes[setName]
  };
```

```
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Esempio di utilizzo delle operazioni di set con AWS SDK for JavaScript.

```
/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
```

```
const config = { region: "us-west-2" };
const tableName = "Users";
const key = { UserId: "U12345" };

console.log("Demonstrating set operations in DynamoDB");

try {
  // Example 1: Create a string set
  console.log("\nExample 1: Creating a string set");
  const response1 = await createSet(
    config,
    tableName,
    key,
    "Interests",
    ["Reading", "Hiking", "Cooking"],
    "string"
  );

  console.log("Created set:", response1.Attributes);

  // Example 2: Add elements to a set
  console.log("\nExample 2: Adding elements to a set");
  const response2 = await addToSet(
    config,
    tableName,
    key,
    "Interests",
    ["Photography", "Travel"],
    "string"
  );

  console.log("Updated set after adding elements:", response2.Attributes);

  // Example 3: Remove elements from a set
  console.log("\nExample 3: Removing elements from a set");
  const response3 = await removeFromSet(
    config,
    tableName,
    key,
    "Interests",
    ["Cooking"],
    "string"
  );
```

```
    console.log("Updated set after removing elements:", response3.Attributes);

    // Example 4: Create a number set
    console.log("\nExample 4: Creating a number set");
    const response4 = await createSet(
      config,
      tableName,
      key,
      "FavoriteNumbers",
      [7, 42, 99],
      "number"
    );

    console.log("Created number set:", response4.Attributes);

    // Example 5: Replace an entire set
    console.log("\nExample 5: Replacing an entire set");
    const response5 = await replaceSet(
      config,
      tableName,
      key,
      "Interests",
      ["Gaming", "Movies", "Music"],
      "string"
    );

    console.log("Replaced set:", response5.Attributes);

    // Example 6: Remove the last element from a set
    console.log("\nExample 6: Removing the last element from a set");

    // First, create a set with just one element
    await createSet(
      config,
      tableName,
      { UserId: "U67890" },
      "Tags",
      ["LastTag"],
      "string"
    );

    // Then, remove the last element
    const response6 = await removeLastElementFromSet(
      config,
```

```
      tableName,
      { UserId: "U67890" },
      "Tags"
    );

    console.log(response6.message);
    console.log("Removed value:", response6.removedValue);

    // Get the final state of the items
    console.log("\nFinal state of the items:");
    const item1 = await getItem(config, tableName, key);
    console.log("User U12345:", JSON.stringify(item1, null, 2));

    const item2 = await getItem(config, tableName, { UserId: "U67890" });
    console.log("User U67890:", JSON.stringify(item2, null, 2));

    // Explain set operations
    console.log("\nKey points about set operations in DynamoDB:");
    console.log("1. Use ADD to add elements to a set (duplicates are automatically
 removed)");
    console.log("2. Use DELETE to remove elements from a set");
    console.log("3. Use SET to create a new set or replace an existing one");
    console.log("4. DynamoDB supports three types of sets: string sets, number sets,
 and binary sets");
    console.log("5. When you delete the last element from a set, the attribute
 remains as an empty set");
    console.log("6. To remove an empty set, use the REMOVE operation");
    console.log("7. Sets automatically maintain unique values (no duplicates)");
    console.log("8. You cannot mix data types within a set");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK for JavaScript API Reference.

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

L'esempio di codice seguente mostra come:

- Ricezione di un batch di elementi mediante più istruzioni SELECT.

- Aggiunta di un batch di articoli eseguendo più istruzioni INSERT.
- Aggiornamento di un batch di elementi mediante più istruzioni UPDATE.
- Eliminazione di un batch di elementi mediante più istruzioni DELETE.

SDK per JavaScript (v3)

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

Eseguire le istruzioni PartiQL in batch.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  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?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({}, { confirmAll });
    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 i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK for JavaScript API Reference.

Esecuzione di una query mediante PartiQL

L'esempio di codice seguente mostra come:

- Ricezione di un articolo eseguendo un'istruzione SELECT.
- Aggiunta di un elemento eseguendo un'istruzione INSERT.
- Aggiornamento di un elemento eseguendo un'istruzione UPDATE.
- Eliminazione di un elemento eseguendo un'istruzione DELETE.

SDK per JavaScript (v3)

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

Eseguire le singole istruzioni PartiQL.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  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?`,
      { type: "confirm", 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 i dettagli sull'API, ExecuteStatementconsulta AWS SDK for JavaScript API Reference.

Eseguire query su una tabella utilizzando un indice secondario globale

L'esempio di codice seguente mostra come eseguire query su una tabella con un indice secondario globale.

- Eseguire query su una tabella DynamoDB utilizzando la relativa chiave primaria.
- Eseguire query su un indice secondario globale (GSI) per modelli di accesso alternativi.
- Confrontare le query su tabelle e le query su indici secondari globali.

SDK per JavaScript (v3)

Interroga una tabella DynamoDB utilizzando la chiave primaria con. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
```

```
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}
```

Interroga un DynamoDB Global Secondary Index (GSI) con. AWS SDK for JavaScript

```
/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);
```

```
    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query su una tabella utilizzando una condizione begins_with

L'esempio di codice seguente mostra come eseguire query su una tabella utilizzando una condizione begins_with.

- Utilizzare la funzione begins_with in un'espressione di condizione chiave.
- Filtrare gli elementi in base a uno schema di prefissi nella chiave di ordinamento.

SDK per (v3) JavaScript

Per eseguire query su una tabella DynamoDB utilizzando una condizione begins_with nella chiave di ordinamento con AWS SDK for JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 prefix
```

```
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */
async function queryWithBeginsWith(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with begins_with: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query su una tabella utilizzando un intervallo di date

L'esempio di codice seguente mostra come eseguire query su una tabella utilizzando un intervallo di date nella chiave di ordinamento.

- Eseguire query sugli elementi all'interno di un intervallo di date specifico.
- Utilizzare gli operatori di confronto nelle chiavi di ordinamento in formato di data.

SDK per (v3 JavaScript)

Interroga una tabella DynamoDB per gli elementi compresi in un intervallo di date con. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
```

```
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query su una tabella con un'espressione di filtro complessa

L'esempio di codice seguente mostra come eseguire query su una tabella con un'espressione di filtro complessa.

- Applicare espressioni di filtro complesse ai risultati delle query.
- Combinare più condizioni utilizzando operatori logici.
- Filtrare gli elementi in base ad attributi non chiave.

SDK per (v3 JavaScript)

Interroga una tabella DynamoDB con un'espressione di filtro complessa utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
```

```
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query su una tabella con un'espressione di filtro dinamica

L'esempio di codice seguente mostra come eseguire query su una tabella con un'espressione di filtro dinamica.

- Creare espressioni di filtro in modo dinamico in fase di runtime.
- Creare condizioni di filtro in base all'input dell'utente o allo stato dell'applicazione.
- Aggiungere o rimuovere i criteri di filtro in modo condizionale.

SDK per (v3 JavaScript)

Interroga una tabella DynamoDB con un'espressione di filtro costruita dinamicamente utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
```

```
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
```

```
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query su una tabella con attributi annidati

L'esempio di codice seguente mostra come eseguire una query su una tabella con attributi annidati.

- Accedere e filtrare in base agli attributi annidati negli elementi DynamoDB.
- Utilizzare le espressioni del percorso del documento per fare riferimento agli elementi annidati.

SDK per (v3) JavaScript

Interroga una tabella DynamoDB con attributi annidati utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
```

```
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
async function queryWithNestedAttribute(
  config,
  tableName,
  productId,
  category
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "product_id = :productId",
      FilterExpression: "details.category = :category",
      ExpressionAttributeValues: {
        ":productId": { S: productId },
        ":category": { S: category }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with nested attribute: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query su una tabella con impaginazione

L'esempio di codice seguente mostra come eseguire una query su una tabella con la paginazione.

- Implementare l'impaginazione per i risultati delle query DynamoDB.
- Utilizzate il LastEvaluatedKey per recuperare le pagine successive.
- Controllare il numero di elementi per pagina con il parametro Limit.

SDK per JavaScript (v3)

Interroga una tabella DynamoDB con impaginazione utilizzando. AWS SDK for JavaScript

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
```

```
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages
```

```
    console.log(`Query complete. Retrieved ${allItems.length} items in ${pageCount}
 pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query su una tabella con elevata consistenza di lettura

L'esempio di codice seguente mostra come eseguire query su una tabella con elevata consistenza di lettura.

- Configurare il livello di consistenza per le query DynamoDB.

- Utilizza letture fortemente coerenti per ottenere la maggior parte dei dati. up-to-date
- Comprende i compromessi tra consistenza effettiva ed elevata consistenza.

SDK per JavaScript (v3)

Interroga una tabella DynamoDB con coerenza di lettura configurabile utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
```

```
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query sui dati utilizzando istruzioni PartiQL SELECT

L'esempio seguente mostra come eseguire query sui dati mediante le istruzioni PartiQL SELECT.

SDK per (v3) JavaScript

Interroga gli elementi da una tabella DynamoDB utilizzando le istruzioni PartiQL SELECT con. AWS SDK for JavaScript

```
/**
 * This example demonstrates how to query items from a DynamoDB table using PartiQL.
 * It shows different ways to select data with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
```

```
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select an item by its primary key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
```

```
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select items using a filter condition with PartiQL.
 *
```

```
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix
 * @param prefix - The prefix to match
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByPrefix = async (
  tableName: string,
  attributeName: string,
  prefix: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE
 begins_with(${attributeName}, ?)`,
    Parameters: [prefix],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a between condition for range queries.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for range
 * @param startValue - The start value of the range
 * @param endValue - The end value of the range
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ? AND ?
`,
    Parameters: [startValue, endValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
```

```
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");

  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456", "productId",
 "prod789");

  // Select with a filter condition (can use any attribute)
  await selectItemsWithFilter("UsersTable", "userType", "premium");

  // Select items with a prefix (useful for hierarchical data)
  await selectItemsByPrefix("ProductsTable", "category", "electronics");

  // Select items within a range (useful for numeric or date ranges)
  await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01", "2023-12-31");
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Eseguire query per elementi TTL

L'esempio di codice seguente mostra come eseguire query su elementi TTL.

SDK per (v3) JavaScript

Interroga l'espressione filtrata per raccogliere elementi TTL in una tabella DynamoDB utilizzando. AWS SDK for JavaScript

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

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1') =>
 {
    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;
    }
}

// Example usage (commented out for testing)
```

```
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Eseguire query utilizzando modelli di data e ora

L'esempio di codice seguente mostra come eseguire query su tabelle utilizzando modelli di data e ora.

- Archivia e interroga date/time i valori in DynamoDB.
- Implementare query basate su intervalli di date utilizzando chiavi di ordinamento.
- Formattare stringhe di data per l'esecuzione efficace di query.

SDK per (v3 JavaScript)

Esegui una query utilizzando intervalli di date nelle chiavi di ordinamento con. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
```

```
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

Interroga utilizzando variabili data-ora con. AWS SDK for JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
```

```
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  dateKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
 AND :endDate`,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#dateAttr": dateKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
```

```
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range: ${error}`);
    throw error;
  }
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Comprendere l'ordine di aggiornamento delle espressioni

L'esempio di codice seguente illustra l'ordine delle espressioni di aggiornamento.

- Informazioni su come DynamoDB elabora le espressioni di aggiornamento.
- Comprendere l'ordine delle operazioni nelle espressioni di aggiornamento.
- Comprendere la valutazione delle espressioni in modo da evitare risultati imprevisti.

SDK per (v3 JavaScript)

Dimostra l'ordine di aggiornamento delle espressioni utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
```

```
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 placeholders
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}
```

```
/**
 * Demonstrate that variables hold copies of existing values before modifications.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that uses the values of attributes before they are modified in the same
 expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateValueCopying(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = { ...key, a: 1, b: 2, c: 3 };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with an expression that uses values before they are
 modified
  // This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to the
 value of 'b'
  const updateResponse = await docClient.send(new UpdateCommand({
```

```
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE a SET b = a, c = b",
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Demonstrate the order in which different action types are processed.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the order
 * in which they are processed.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
counter :increment DELETE set_attr :elements",
    ExpressionAttributeValues: {
      ":new_value": "Updated value",
      ":increment": 5,
      ":elements": new Set(["B"])
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));
```

```
  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Update multiple attributes with a single SET action.
 *
 * This function demonstrates how to update multiple attributes in a single SET
 action,
 * which is more efficient than using multiple separate update operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {Object} attributes - The attributes to update and their new values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleSetActions(
  config,
  tableName,
  key,
  attributes
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET ";
  const expressionAttributeValues = {};

  // Add each attribute to the update expression
  Object.entries(attributes).forEach(([attrName, attrValue], index) => {
    const valuePlaceholder = `:val${index}`;

    if (index > 0) {
      updateExpression += ", ";
    }
```

```
    updateExpression += `${attrName} = ${valuePlaceholder}`;

    expressionAttributeValues[valuePlaceholder] = attrValue;
  });

  // Execute the update
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Update an attribute with a value from another attribute or a default value.
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a value
 * from one attribute to another, or use a default value if the source doesn't
 exist.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} sourceAttribute - The attribute to copy the value from
 * @param {string} targetAttribute - The attribute to update
 * @param {any} defaultValue - The default value to use if the source attribute
 doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
 if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Demonstrate complex update expressions with multiple operations on the same
 attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
 attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    list_attr: [1, 2, 3],
```

```
  map_attr: {
    nested1: "value1",
    nested2: "value2"
  }
};

await docClient.send(new PutCommand({
  TableName: tableName,
  Item: initialItem
}));

// Step 2: Get the item to verify initial state
const responseBefore = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemBefore = responseBefore.Item || {};

// Step 3: Update the item with multiple operations on the same attributes
const updateResponse = await docClient.send(new UpdateCommand({
  TableName: tableName,
  Key: key,
  UpdateExpression: `
    SET counter = counter + :inc1,
        counter = counter + :inc2,
        map_attr.nested1 = :new_val1,
        map_attr.nested3 = :new_val3,
        list_attr[0] = list_attr[1],
        list_attr[1] = list_attr[2]
  `,
  ExpressionAttributeValues: {
    ":inc1": 5,
    ":inc2": 3,
    ":new_val1": "updated_value1",
    ":new_val3": "new_value3"
  },
  ReturnValues: "UPDATED_NEW"
}));

// Step 4: Get the item to verify final state
const responseAfter = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
```

```
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}
```

Esempio di utilizzo dell'ordine di aggiornamento delle espressioni con AWS SDK for JavaScript.

```
/**
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "OrderProcessing";

  console.log("Demonstrating update expression order of operations in DynamoDB");

  try {
    // Example 1: Demonstrating value copying in update expressions
    console.log("\nExample 1: Demonstrating value copying in update expressions");
    const results1 = await demonstrateValueCopying(
      config,
      tableName,
      { OrderId: "order123" }
    );

    console.log("Initial state:", JSON.stringify(results1.initialState, null, 2));
    console.log("Update response:", JSON.stringify(results1.updateResponse, null,
 2));
    console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The initial state had a=1, b=2, c=3");
    console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
 following:");
```

```
   console.log("   - Copied the value of 'a' (which was 1) to be used for 'b'");
   console.log("   - Copied the value of 'b' (which was 2) to be used for 'c'");
   console.log("   - Removed the attribute 'a'");
   console.log("3. The final state has b=1, c=2, and 'a' is removed");
   console.log("4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications");

   // Example 2: Demonstrating the order of different action types
   console.log("\nExample 2: Demonstrating the order of different action types");
   const results2 = await demonstrateActionOrder(
     config,
     tableName,
     { OrderId: "order456" }
   );

   console.log("Initial state:", JSON.stringify(results2.initialState, null, 2));
   console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE");
   console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
   console.log("3. First, 'to_remove' was removed");
   console.log("4. Then, 'to_modify' was set to a new value");
   console.log("5. Next, 'counter' was incremented by 5");
   console.log("6. Finally, 'B' was removed from the set attribute");

   // Example 3: Updating multiple attributes in a single SET action
   console.log("\nExample 3: Updating multiple attributes in a single SET action");
   const response3 = await updateWithMultipleSetActions(
     config,
     tableName,
     { OrderId: "order789" },
     {
       Status: "Shipped",
       ShippingDate: "2025-05-28",
       TrackingNumber: "1Z999AA10123456784"
     }
   );
```

```
   console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

   // Example 4: Conditional value copying with if_not_exists
   console.log("\nExample 4: Conditional value copying with if_not_exists");
   const response4 = await updateWithConditionalValueCopying(
     config,
     tableName,
     { OrderId: "order101" },
     "PreferredShippingMethod",
     "ShippingMethod",
     "Standard"
   );

   console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

   // Example 5: Multiple operations on the same attribute
   console.log("\nExample 5: Multiple operations on the same attribute");
   const results5 = await demonstrateMultipleOperationsOnSameAttribute(
     config,
     tableName,
     { OrderId: "order202" }
   );

   console.log("Initial state:", JSON.stringify(results5.initialState, null, 2));
   console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results5.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The counter was incremented twice (first by 5, then by 3) for a
total of +8");
   console.log("2. The map attribute had one value updated and a new nested
attribute added");
   console.log("3. The list attribute had values shifted (value at index 1 moved to
index 0, value at index 2 moved to index 1)");
   console.log("4. All operations within the SET action are processed from left to
right");

   // Key points about update expression order of operations
   console.log("\nKey Points About Update Expression Order of Operations:");
   console.log("1. Variables in expressions hold copies of attribute values as they
existed BEFORE any modifications");
```

```
    console.log("2. Multiple actions in an update expression are processed in this
 order: REMOVE, SET, ADD, DELETE");
    console.log("3. Within each action type, operations are processed from left to
 right");
    console.log("4. You can reference the same attribute multiple times in an
 expression");
    console.log("5. You can use if_not_exists() to conditionally set values based on
 attribute existence");
    console.log("6. Using a single update expression with multiple actions is more
 efficient than multiple separate updates");
    console.log("7. The update expression is atomic - either all actions succeed or
 none do");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK for JavaScript API Reference.

Aggiornare l'impostazione del throughput a caldo di una tabella

L'esempio di codice seguente mostra come aggiornare l'impostazione del throughput a caldo di una tabella.

SDK per JavaScript (v3)

Aggiornare l'impostazione del throughput a caldo di tabella DynamoDB con un SDK AWS SDK for JavaScript.

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

export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
```

```
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response:
 ${JSON.stringify(response)}`);
    return response;
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
updateDynamoDBTableWarmThroughput(
  'example-table',
  5, 5,
  'example-index',
  2, 2
);
*/
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API UpdateTableReference.

Aggiornare il TTL di un elemento

L'esempio di codice seguente mostra come aggiornare il valore TTL di un elemento.

SDK per JavaScript (v3)

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

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    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;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-value');
```

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

Aggiorna i dati utilizzando PartiQL UPDATE

L'esempio di codice seguente mostra come aggiornare i dati mediante le istruzioni PartiQL UPDATE.

SDK per JavaScript (v3)

Aggiorna gli elementi in una tabella DynamoDB utilizzando le istruzioni PartiQL UPDATE con. AWS SDK for JavaScript

```
/**
 * This example demonstrates how to update items in a DynamoDB table using PartiQL.
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateSingleAttribute = async (
```

```
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update multiple attributes of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeUpdates - Object containing attribute names and their new values
 * @returns The response from the ExecuteStatementCommand
 */
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
```

```
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");

  // Create parameters array with attribute values followed by the partition key
 value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName} = ?
`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item identified by a composite key (partition key + sort key) using
 PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
```

```
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item with a condition to ensure the update only happens if a condition
 is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
```

```
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
 * Batch update multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param updates - Array of objects containing key and update information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each update
  const statements = updates.map((update) => {
    return {
      Statement: `UPDATE "${tableName}" SET ${update.attributeName} = ? WHERE
 ${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
```

```
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
  await updateSingleAttribute("UsersTable", "userId", "user123", "email",
 "newemail@example.com");

  // Update multiple attributes at once
  await updateMultipleAttributes("UsersTable", "userId", "user123", {
    email: "newemail@example.com",
    name: "John Smith",
    lastLogin: new Date().toISOString(),
  });

  // Update an item with a composite key (partition key + sort key)
  await updateItemWithCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789",
    "quantity",
    5
  );

  // Update with a condition
  await updateItemWithCondition(
```

```
    "UsersTable",
    "userId",
    "user123",
    "userStatus",
    "active",
    "userType",
    "premium"
  );

  // Batch update multiple items
  await batchUpdateItems("UsersTable", [
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user123",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user456",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
  ]);
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Utilizzo di Gateway API per invocare una funzione Lambda

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

SDK per JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando l'API di JavaScript runtime Lambda. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo

esempio viene illustrato come creare una funzione Lambda invocata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

Utilizzare le operazioni dei contatori atomici

L'esempio di codice seguente mostra come utilizzare le operazioni dei contatori atomici in DynamoDB.

- Incrementare i contatori atomicamente utilizzando le operazioni ADD e SET.
- Incrementare in modo sicuro i contatori eventualmente inesistenti.
- Implementare un blocco ottimistico per le operazioni dei contatori.

SDK per JavaScript (v3)

Dimostra le controoperazioni atomiche utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
```

```
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 increments.
```

```
 * While this approach works, it's less idiomatic for simple increments than using
 ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with an expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  defaultValue = 0
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = if_not_exists(${counterName}, :default)
 + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement optimistic
 * locking, which prevents race conditions when multiple processes try to update
```

```
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithLocking(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  expectedValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ConditionExpression: `${counterName} = :expected`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":expected": expectedValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      data: response
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
```

```
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        error: "Optimistic locking failed: the counter value has changed"
      };
    }
    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not found
 */
async function getCounterValue(
  config,
  tableName,
  key,
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
```

```
    : null;
}
```

Esempio di utilizzo delle operazioni di conteggio atomico con. AWS SDK for JavaScript

```
/**
 * Example of how to use the atomic counter operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;

  console.log("Demonstrating different approaches to increment counters in
 DynamoDB");

  try {
    // Example 1: Using ADD operation (recommended for simple increments)
    console.log("\nExample 1: Incrementing counter with ADD operation");
    const response1 = await incrementCounterWithAdd(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);

    // Example 2: Using SET operation with an expression
    console.log("\nExample 2: Incrementing counter with SET operation");
    const response2 = await incrementCounterWithSet(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );
```

```
    console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

    // Example 3: Safely incrementing a counter that might not exist
    console.log("\nExample 3: Safely incrementing counter that might not exist");
    const newKey = { ProductId: "P67890" };
    const response3 = await incrementCounterSafely(
      config,
      tableName,
      newKey,
      counterName,
      incrementValue,
      0
    );

    console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

    // Example 4: Incrementing with optimistic locking
    console.log("\nExample 4: Incrementing with optimistic locking");

    // First, get the current counter value
    const currentValue = await getCounterValue(config, tableName, key, counterName);
    console.log(`Current counter value: ${currentValue}`);

    // Then, try to increment with optimistic locking
    const response4 = await incrementCounterWithLocking(
      config,
      tableName,
      key,
      counterName,
      incrementValue,
      currentValue
    );

    if (response4.success) {
      console.log(`Counter successfully incremented to:
${response4.data.Attributes[counterName]}`);
    } else {
      console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
```

```
    console.log("2. SET with expressions is more flexible for complex operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might not
 exist");
    console.log("5. Optimistic locking can be added to either approach for
 additional safety");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK for JavaScript API Reference.

Utilizzare le operazioni condizionali

L'esempio di codice seguente mostra come utilizzare le operazioni condizionali in DynamoDB.

- Implementare scritture condizionali per impedire la sovrascrittura dei dati.
- Utilizzare le espressioni condizionali per applicare le regole aziendali.
- Gestire gli errori dei controlli condizionali con gradualità.

SDK per JavaScript (v3)

Dimostra le operazioni condizionali utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  DeleteCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue,
  updateAttribute,
  updateValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateAttribute} = :value`,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":value": updateValue,
      ":condition": conditionValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Condition was met and update was performed",
      updatedAttributes: response.Attributes
    };
```

```
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, update was not performed",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Perform a conditional delete operation.
 *
 * This function demonstrates how to delete an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to delete
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalDelete(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the delete parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
```

```
      ":condition": conditionValue
    },
    ReturnValues: "ALL_OLD"
  };

  try {
    // Perform the delete operation
    const response = await docClient.send(new DeleteCommand(params));

    return {
      success: true,
      message: "Condition was met and item was deleted",
      deletedItem: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, item was not deleted",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
 * to prevent race conditions when multiple processes update the same item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {Object} updates - The attributes to update
 * @param {number} expectedVersion - The expected current version number
 * @returns {Promise<Object>} - Result of the operation
 */
async function updateWithOptimisticLocking(
  config,
```

```
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Add the version update
  updateExpressions.push("version = :newVersion");

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Update succeeded with optimistic locking",
      newVersion: expectedVersion + 1,
      updatedAttributes: response.Attributes
```

```
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Optimistic locking failed: the item was modified by another
 process",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 operation).
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
async function createIfNotExists(
  config,
  tableName,
  item
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Extract the primary key attributes
  const keyAttributes = Object.keys(item).filter(attr =>
    attr === "id" || attr === "ID" || attr === "Id" ||
    attr.endsWith("Id") || attr.endsWith("ID") ||
    attr.endsWith("Key")
  );
```

```
  if (keyAttributes.length === 0) {
    throw new Error("Could not determine primary key attributes");
  }

  // Create a condition expression that checks if the item doesn't exist
  const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

  // Define the put parameters with a condition expression
  const params = {
    TableName: tableName,
    Item: item,
    ConditionExpression: conditionExpression
  };

  try {
    // Perform the put operation
    await docClient.send(new PutCommand(params));

    return {
      success: true,
      message: "Item was created because it didn't exist",
      item
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Item already exists, creation was skipped",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Esempio di utilizzo di operazioni condizionali con. AWS SDK for JavaScript

```
/**
 * Example of how to use conditional operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating conditional operations in DynamoDB");

  try {
    // Example 1: Conditional update based on attribute value
```

```
  console.log("\nExample 1: Conditional update based on attribute value");
  const updateResult = await conditionalUpdate(
    config,
    tableName,
    key,
    "Category",
    "Electronics",
    "Price",
    299.99
  );

  console.log(`Result: ${updateResult.message}`);
  if (updateResult.success) {
    console.log("Updated attributes:", updateResult.updatedAttributes);
  }

  // Example 2: Conditional delete based on attribute value
  console.log("\nExample 2: Conditional delete based on attribute value");
  const deleteResult = await conditionalDelete(
    config,
    tableName,
    key,
    "InStock",
    false
  );

  console.log(`Result: ${deleteResult.message}`);
  if (deleteResult.success) {
    console.log("Deleted item:", deleteResult.deletedItem);
  }

  // Example 3: Optimistic locking with version number
  console.log("\nExample 3: Optimistic locking with version number");

  // First, get the current item to check its version
  const currentItem = await getItem(config, tableName, { ProductId: "P67890" });
  const currentVersion = currentItem ? (currentItem.version || 0) : 0;

  console.log(`Current version: ${currentVersion}`);

  // Then, update with optimistic locking
  const lockingResult = await updateWithOptimisticLocking(
    config,
    tableName,
```

```
      { ProductId: "P67890" },
      {
        Name: "Updated Product Name",
        Description: "This is an updated description"
      },
      currentVersion
    );

    console.log(`Result: ${lockingResult.message}`);
    if (lockingResult.success) {
      console.log(`New version: ${lockingResult.newVersion}`);
      console.log("Updated attributes:", lockingResult.updatedAttributes);
    }

    // Example 4: Create item only if it doesn't exist
    console.log("\nExample 4: Create item only if it doesn't exist");
    const createResult = await createIfNotExists(
      config,
      tableName,
      {
        ProductId: "P99999",
        Name: "New Product",
        Category: "Accessories",
        Price: 19.99,
        InStock: true
      }
    );

    console.log(`Result: ${createResult.message}`);
    if (createResult.success) {
      console.log("Created item:", createResult.item);
    }

    // Explain conditional operations
    console.log("\nKey points about conditional operations:");
    console.log("1. Conditional operations only succeed if the condition is met");
    console.log("2. ConditionalCheckFailedException indicates the condition wasn't
met");
    console.log("3. Optimistic locking prevents race conditions in concurrent
updates");
    console.log("4. attribute_exists and attribute_not_exists are useful for
checking if attributes are present");
    console.log("5. Conditional operations are atomic - they either succeed
completely or fail completely");
```

```
    console.log("6. You can use any valid comparison operators and functions in
 condition expressions");
    console.log("7. Conditional operations don't consume write capacity if the
 condition fails");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - DeleteItem
  - PutItem
  - UpdateItem

Utilizzare i nomi degli attributi di espressione

L'esempio di codice seguente mostra come utilizzare i nomi degli attributi di espressione in DynamoDB.

- Utilizzare le parole riservate nelle espressioni DynamoDB.
- Utilizzare i segnaposto per i nomi degli attributi di espressione.
- Gestire i caratteri speciali nei nomi degli attributi.

SDK per JavaScript (v3)

Dimostra i nomi degli attributi di espressione utilizzando. AWS SDK for JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  QueryCommand,
  ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
```

```
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateReservedWordAttribute(
  config,
  tableName,
  key,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
```

```
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters to
 update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
  specialCharAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": specialCharAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}
```

```
/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
 * @param {string} reservedWordAttribute - The reserved word attribute to filter on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the query parameters using expression attribute names
  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pkName = :pkValue",
    FilterExpression: "#attr = :value",
    ExpressionAttributeNames: {
      "#pkName": partitionKeyName,
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":pkValue": partitionKeyValue,
      ":value": value
    }
  };

  // Perform the query operation
  const response = await docClient.send(new QueryCommand(params));
```

```
  return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedReservedWordAttribute(
  config,
  tableName,
  key,
  attributePath,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names for each part of the path
  const expressionAttributeNames = {};
  for (let i = 0; i < attributePath.length; i++) {
    expressionAttributeNames[`#attr${i}`] = attributePath[i];
  }

  // Build the attribute path using the expression attribute names
  const attributePathExpression = attributePath
    .map((_, i) => `#attr${i}`)
    .join(".");

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
```

```
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
 * in a complex filter expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function scanWithMultipleAttributeNames(
  config,
  tableName,
  filters
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names and values
  const expressionAttributeNames = {};
  const expressionAttributeValues = {};
  const filterConditions = [];

  // Build the filter expression
  Object.entries(filters).forEach(([attrName, value], index) => {
    const nameKey = `#attr${index}`;
    const valueKey = `:val${index}`;

    expressionAttributeNames[nameKey] = attrName;
    expressionAttributeValues[valueKey] = value;
```

```
    filterConditions.push(`${nameKey} = ${valueKey}`);
  });

  // Join the filter conditions with AND
  const filterExpression = filterConditions.join(" AND ");

  // Define the scan parameters
  const params = {
    TableName: tableName,
    FilterExpression: filterExpression,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Perform the scan operation
  const response = await docClient.send(new ScanCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };
```

```
  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Esempio di utilizzo dei nomi degli attributi di espressione con AWS SDK for JavaScript.

```
/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating expression attribute names in DynamoDB");

  try {
    // Example 1: Update an attribute that is a reserved word
    console.log("\nExample 1: Updating an attribute that is a reserved word");
    const response1 = await updateReservedWordAttribute(
      config,
      tableName,
      key,
      "Size", // "SIZE" is a reserved word in DynamoDB
      "Large"
    );

    console.log("Updated attribute:", response1.Attributes);

    // Example 2: Update an attribute with special characters
    console.log("\nExample 2: Updating an attribute with special characters");
    const response2 = await updateSpecialCharacterAttribute(
      config,
      tableName,
      key,
      "Product-Type", // Contains a hyphen, which is a special character
      "Electronics"
```

```
    );

    console.log("Updated attribute:", response2.Attributes);

    // Example 3: Query with a reserved word attribute
    console.log("\nExample 3: Querying with a reserved word attribute");
    const response3 = await queryWithReservedWordAttribute(
      config,
      tableName,
      "Category",
      "Electronics",
      "Count", // "COUNT" is a reserved word in DynamoDB
      10
    );

    console.log(`Found ${response3.Items.length} items`);

    // Example 4: Update a nested attribute with reserved words in the path
    console.log("\nExample 4: Updating a nested attribute with reserved words in the
path");
    const response4 = await updateNestedReservedWordAttribute(
      config,
      tableName,
      key,
      ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
      30
    );

    console.log("Updated nested attribute:", response4.Attributes);

    // Example 5: Scan with multiple attribute name placeholders
    console.log("\nExample 5: Scanning with multiple attribute name placeholders");
    const response5 = await scanWithMultipleAttributeNames(
      config,
      tableName,
      {
        "Size": "Large",
        "Count": 10,
        "Product-Type": "Electronics"
      }
    );

    console.log(`Found ${response5.Items.length} items`);
```

```
    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Show some common reserved words
    console.log("\nSome common DynamoDB reserved words:");
    const commonReservedWords = [
      "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
      "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
      "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
      "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
      "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN", "NAME",
      "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
      "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
      "VIEW", "WHERE"
    ];
    console.log(commonReservedWords.join(", "));

    // Explain expression attribute names
    console.log("\nKey points about expression attribute names:");
    console.log("1. Use expression attribute names (#name) for reserved words");
    console.log("2. Use expression attribute names for attributes with special
 characters");
    console.log("3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
    console.log("4. Expression attribute names are required for nested attributes
 with reserved words");
    console.log("5. You can use multiple expression attribute names in a single
 expression");
    console.log("6. Expression attribute names are case-sensitive");
    console.log("7. Expression attribute names are only used in expressions, not in
 the actual data");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - Query

- UpdateItem

Utilizzo degli eventi pianificati per invocare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

SDK per JavaScript (v3)

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Esempi serverless

Invocare una funzione Lambda da un trigger DynamoDB

L'esempio di codice seguente 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.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. 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 di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

L'esempio di codice seguente mostra come implementare una risposta batch parziale per 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.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. 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 };
};
```

EC2 Esempi di Amazon che utilizzano SDK per JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon EC2.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base

- Azioni
- Scenari

Nozioni di base

Ciao Amazon EC2

Il seguente esempio di codice mostra come iniziare a usare Amazon EC2.

SDK per JavaScript (v3)

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

```
import { DescribeSecurityGroupsCommand, 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 "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Per i dettagli sull'API, DescribeSecurityGroupsconsulta AWS SDK for JavaScript API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Creare una coppia di chiavi e un gruppo di sicurezza.
- Selezionare un'Amazon Machine Image (AMI) e un tipo di istanza compatibile e quindi creare un'istanza.
- Arrestare e riavviare l'istanza.
- Associazione di un indirizzo IP elastico all'istanza.
- Connettiti alla tua istanza con SSH, quindi elimina le risorse.

SDK per JavaScript (v3)

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

Questo file contiene un elenco di azioni comuni utilizzate con EC2. I passaggi sono creati in base a un framework di 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) {
        for (const param of page.Parameters) {
          // 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.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 informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - 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 usare`AllocateAddress`.

SDK per JavaScript (v3)

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

```
import { 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 "node:url";
// Call function if run directly.
```

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

- Per i dettagli sull'API, AllocateAddressconsulta AWS SDK for JavaScript API Reference.

`AssociateAddress`

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

SDK per JavaScript (v3)

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

```
import { AssociateAddressCommand, 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 i dettagli sull'API, AssociateAddressconsulta AWS SDK for JavaScript API Reference.

`AuthorizeSecurityGroupIngress`

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

SDK per JavaScript (v3)

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

```
import {
  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 i dettagli sull'API, AuthorizeSecurityGroupIngressconsulta AWS SDK for JavaScript API Reference.

`CreateKeyPair`

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

SDK per JavaScript (v3)

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

```
import { CreateKeyPairCommand, 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 i dettagli sull'API, CreateKeyPairconsulta AWS SDK for JavaScript API Reference.

`CreateLaunchTemplate`

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

SDK per JavaScript (v3)

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

```
const ssmClient = new SSMClient({});
const { Parameter } = await ssmClient.send(
  new GetParameterCommand({
    Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
  }),
);
const ec2Client = new EC2Client({});
await ec2Client.send(
  new CreateLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
      UserData: readFileSync(
        join(RESOURCES_PATH, "server_startup_script.sh"),
      ).toString("base64"),
      KeyName: NAMES.keyPairName,
    },
  }),
```

- Per i dettagli sull'API, CreateLaunchTemplateconsulta AWS SDK for JavaScript API Reference.

`CreateSecurityGroup`

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

SDK per JavaScript (v3)

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

```
import { CreateSecurityGroupCommand, 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 i dettagli sull'API, CreateSecurityGroupconsulta AWS SDK for JavaScript API Reference.

`DeleteKeyPair`

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

SDK per JavaScript (v3)

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

```
import { DeleteKeyPairCommand, 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 i dettagli sull'API, DeleteKeyPairconsulta AWS SDK for JavaScript API Reference.

`DeleteLaunchTemplate`

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

SDK per JavaScript (v3)

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

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- Per i dettagli sull'API, DeleteLaunchTemplateconsulta AWS SDK for JavaScript API Reference.

`DeleteSecurityGroup`

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

SDK per JavaScript (v3)

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

```
import { DeleteSecurityGroupCommand, 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 i dettagli sull'API, DeleteSecurityGroupconsulta AWS SDK for JavaScript API Reference.

`DescribeAddresses`

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

SDK per JavaScript (v3)

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

```
import { DescribeAddressesCommand, 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 i dettagli sull'API, DescribeAddressesconsulta AWS SDK for JavaScript API Reference.

`DescribeIamInstanceProfileAssociations`

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

SDK per JavaScript (v3)

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

```
    const ec2Client = new EC2Client({});
    const { IamInstanceProfileAssociations } = await ec2Client.send(
      new DescribeIamInstanceProfileAssociationsCommand({
        Filters: [
          { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
        ],
      }),
```

```
    );
```

- Per i dettagli sull'API, DescribeIamInstanceProfileAssociationsconsulta AWS SDK for JavaScript API Reference.

`DescribeImages`

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

SDK per JavaScript (v3)

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

```
import { 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 = Number.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;
      }
      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 i dettagli sull'API, DescribeImagesconsulta AWS SDK for JavaScript API Reference.

`DescribeInstanceTypes`

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

SDK per JavaScript (v3)

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

```
import { 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 = Number.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 i dettagli sull'API, DescribeInstanceTypesconsulta AWS SDK for JavaScript API Reference.

`DescribeInstances`

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

SDK per JavaScript (v3)

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

```
import { 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 = Number.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;
      for (const reservation of Reservations) {
        instanceList.push(...reservation.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 i dettagli sull'API, DescribeInstancesconsulta AWS SDK for JavaScript API Reference.

`DescribeKeyPairs`

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

SDK per JavaScript (v3)

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

```
import { DescribeKeyPairsCommand, 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 i dettagli sull'API, DescribeKeyPairsconsulta AWS SDK for JavaScript API Reference.

`DescribeRegions`

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

SDK per JavaScript (v3)

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

```
import { DescribeRegionsCommand, 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 i dettagli sull'API, DescribeRegionsconsulta AWS SDK for JavaScript API Reference.

`DescribeSecurityGroups`

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

SDK per JavaScript (v3)

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

```
import { DescribeSecurityGroupsCommand, 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 i dettagli sull'API, DescribeSecurityGroupsconsulta AWS SDK for JavaScript API Reference.

`DescribeSubnets`

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

SDK per JavaScript (v3)

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

```
const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  }),
);
```

- Per i dettagli sull'API, DescribeSubnetsconsulta AWS SDK for JavaScript API Reference.

`DescribeVpcs`

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

SDK per JavaScript (v3)

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

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
```

```
    );
```

- Per i dettagli sull'API, DescribeVpcsconsulta AWS SDK for JavaScript API Reference.

`DisassociateAddress`

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

SDK per JavaScript (v3)

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

```
import { 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 i dettagli sull'API, DisassociateAddressconsulta AWS SDK for JavaScript API Reference.

`MonitorInstances`

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

SDK per JavaScript (v3)

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

```
import { 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 i dettagli sull'API, MonitorInstancesconsulta AWS SDK for JavaScript API Reference.

`RebootInstances`

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

SDK per JavaScript (v3)

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

```
import { 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 i dettagli sull'API, RebootInstancesconsulta AWS SDK for JavaScript API Reference.

`ReleaseAddress`

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

SDK per JavaScript (v3)

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

```
import { ReleaseAddressCommand, 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 i dettagli sull'API, ReleaseAddressconsulta AWS SDK for JavaScript API Reference.

`ReplaceIamInstanceProfileAssociation`

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

SDK per JavaScript (v3)

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

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );
```

- Per i dettagli sull'API, ReplaceIamInstanceProfileAssociationconsulta AWS SDK for JavaScript API Reference.

`RunInstances`

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

SDK per JavaScript (v3)

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

```
import { 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 = Number.parseInt(minCount);
  maxCount = Number.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 i dettagli sull'API, RunInstancesconsulta AWS SDK for JavaScript API Reference.

`StartInstances`

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

SDK per JavaScript (v3)

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

```
import { EC2Client, StartInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node: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 i dettagli sull'API, StartInstancesconsulta AWS SDK for JavaScript API Reference.

`StopInstances`

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

SDK per JavaScript (v3)

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

```
import { EC2Client, StopInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node: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 i dettagli sull'API, StopInstancesconsulta AWS SDK for JavaScript API Reference.

`TerminateInstances`

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

SDK per JavaScript (v3)

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

```
import { EC2Client, TerminateInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node: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 i dettagli sull'API, TerminateInstancesconsulta AWS SDK for JavaScript API Reference.

`UnmonitorInstances`

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

SDK per JavaScript (v3)

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

```
import { EC2Client, UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node: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 i dettagli sull'API, UnmonitorInstancesconsulta AWS SDK for JavaScript API Reference.

Scenari

Creazione e gestione di un servizio resiliente

L'esempio di codice seguente 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 (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze in un intervallo specificato.
- Gestisci e distribuisci le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo Auto Scaling 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.

SDK per JavaScript (v3)

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

Esegui lo scenario interattivo al prompt dei comandi.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
```

```
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-
v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}
```

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),
        );
```

```
      }
      return MESSAGES.noIpRules;
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      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);
    }
```

```
    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(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${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,
      );
    }
    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);
    }
    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,
      );
    }
    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);
    }
    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(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
    return MESSAGES.deletedInstanceRole.replace(
      "${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(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
    return MESSAGES.deletedInstanceProfile.replace(
      "${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,
      );
    }
    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,
      );
    }
    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(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
  return MESSAGES.deletedLoadBalancer.replace(
    "${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,
    );
```

```
    }
    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("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${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);
    }
    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");
    }
    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(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${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(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
    return MESSAGES.deletedSsmOnlyPolicy.replace(
      "${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(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${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(
        "${IP}",
        state.myIp,
      );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${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;
    }
    console.log(err.name);
```

```
    throw err;
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Elastic Load Balancing - Esempi della versione 2 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 Elastic Load Balancing - Version 2.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Hello Elastic Load Balancing

Il seguente esempio di codice mostra come iniziare a utilizzare Elastic Load Balancing.

SDK per JavaScript (v3)

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

```
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 "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Per i dettagli sull'API, DescribeLoadBalancersconsulta AWS SDK for JavaScript API Reference.

Azioni

`CreateListener`

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

SDK per JavaScript (v3)

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

```
    const client = new ElasticLoadBalancingV2Client({});
```

```
  const { Listeners } = await client.send(
    new CreateListenerCommand({
      LoadBalancerArn: state.loadBalancerArn,
      Protocol: state.targetGroupProtocol,
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  );
```

- Per i dettagli sull'API, CreateListenerconsulta AWS SDK for JavaScript API Reference.

`CreateLoadBalancer`

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

SDK per JavaScript (v3)

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

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new CreateLoadBalancerCommand({
    Name: NAMES.loadBalancerName,
    Subnets: state.subnets,
  }),
);
state.loadBalancerDns = LoadBalancers[0].DNSName;
state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
await waitUntilLoadBalancerAvailable(
  { client },
  { Names: [NAMES.loadBalancerName] },
);
```

- Per i dettagli sull'API, CreateLoadBalancerconsulta AWS SDK for JavaScript API Reference.

`CreateTargetGroup`

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

SDK per JavaScript (v3)

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

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new CreateTargetGroupCommand({
    Name: NAMES.loadBalancerTargetGroupName,
    Protocol: "HTTP",
    Port: 80,
    HealthCheckPath: "/healthcheck",
    HealthCheckIntervalSeconds: 10,
    HealthCheckTimeoutSeconds: 5,
    HealthyThresholdCount: 2,
    UnhealthyThresholdCount: 2,
    VpcId: state.defaultVpc,
  }),
);
```

- Per i dettagli sull'API, CreateTargetGroupconsulta AWS SDK for JavaScript API Reference.

`DeleteLoadBalancer`

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

SDK per JavaScript (v3)

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

```
  const client = new ElasticLoadBalancingV2Client({});
  const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
  await client.send(
    new DeleteLoadBalancerCommand({
      LoadBalancerArn: loadBalancer.LoadBalancerArn,
    }),
  );
  await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
    const lb = await findLoadBalancer(NAMES.loadBalancerName);
    if (lb) {
      throw new Error("Load balancer still exists.");
    }
  });
```

- Per i dettagli sull'API, DeleteLoadBalancerconsulta AWS SDK for JavaScript API Reference.

`DeleteTargetGroup`

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

SDK per JavaScript (v3)

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

```
  const client = new ElasticLoadBalancingV2Client({});
  try {
```

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

    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      client.send(
        new DeleteTargetGroupCommand({
          TargetGroupArn: TargetGroups[0].TargetGroupArn,
        }),
      ),
    );
  } catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
  }
```

- Per i dettagli sull'API, DeleteTargetGroupconsulta AWS SDK for JavaScript API Reference.

`DescribeLoadBalancers`

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

SDK per JavaScript (v3)

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

```
import {
  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 "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Per i dettagli sull'API, DescribeLoadBalancersconsulta AWS SDK for JavaScript API Reference.

`DescribeTargetGroups`

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

SDK per JavaScript (v3)

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

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

- Per i dettagli sull'API, DescribeTargetGroupsconsulta AWS SDK for JavaScript API Reference.

`DescribeTargetHealth`

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

SDK per JavaScript (v3)

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

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

- Per i dettagli sull'API, DescribeTargetHealthconsulta AWS SDK for JavaScript API Reference.

Scenari

Creazione e gestione di un servizio resiliente

L’esempio di codice seguente 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 (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze in un intervallo specificato.
- Gestisci e distribuisci le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo Auto Scaling 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.

SDK per JavaScript (v3)

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

Esegui lo scenario interattivo al prompt dei comandi.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
```

```
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-
v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}
```

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),
        );
      }
      return MESSAGES.noIpRules;
    },
```

```
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      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);
    }
    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(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${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,
      );
    }
    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);
    }
    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,
      );
    }
    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);
    }
    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(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
    return MESSAGES.deletedInstanceRole.replace(
      "${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(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
    return MESSAGES.deletedInstanceProfile.replace(
      "${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(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${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(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${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(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${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(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${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);
  }
  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);
  }
  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");
  }
  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(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${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(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
    return MESSAGES.deletedSsmOnlyPolicy.replace(
      "${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(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${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(
        "${IP}",
        state.myIp,
      );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${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;
    }
    console.log(err.name);
    throw err;
  }
}
```

```
/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups

- CreateAutoScalingGroup
- CreateInstanceProfile
- CreateLaunchTemplate
- CreateListener
- CreateLoadBalancer
- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

AWS Entity Resolution 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 Entity Resolution

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Salve AWS Entity Resolution

L'esempio di codice seguente mostra come iniziare a utilizzare AWS Entity Resolution.

SDK per JavaScript (v3)

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

```
import { fileURLToPath } from "node:url";
import {
  EntityResolutionClient,
  ListMatchingWorkflowsCommand,
} from "@aws-sdk/client-entityresolution";

export const main = async () => {
  const region = "eu-west-1";
  const erClient = new EntityResolutionClient({ region: region });
  try {
    const command = new ListMatchingWorkflowsCommand({});
    const response = await erClient.send(command);
    const workflowSummaries = response.workflowSummaries;
    for (const workflowSummary of workflowSummaries) {
      console.log(`Attribute name: ${workflowSummaries[0].workflowName} `);
```

```
    }
    if (workflowSummaries.length === 0) {
      console.log("No matching workflows found.");
    }
  } catch (error) {
    console.error(
      `An error occurred in listing the workflow summaries: ${error.message} \n
 Exiting program.`,
    );
    return;
  }
};
```

- Per i dettagli sull'API, ListMatchingWorkflowsconsulta AWS SDK for JavaScript API Reference.

Azioni

`CreateMatchingWorkflow`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  CreateMatchingWorkflowCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };
```

```
const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const createMatchingWorkflowParams = {
    roleArn: `${data.inputs.roleArn}`,
    workflowName: `${data.inputs.workflowName}`,
    description: "Created by using the AWS SDK for JavaScript (v3).",
    inputSourceConfig: [
      {
        inputSourceARN: `${data.inputs.JSONinputSourceARN}`,
        schemaName: `${data.inputs.schemaNameJson}`,
        applyNormalization: false,
      },
      {
        inputSourceARN: `${data.inputs.CSVinputSourceARN}`,
        schemaName: `${data.inputs.schemaNameCSV}`,
        applyNormalization: false,
      },
    ],
    outputSourceConfig: [
      {
        outputS3Path: `s3://${data.inputs.myBucketName}/eroutput`,
        output: [
          {
            name: "id",
          },
          {
            name: "name",
          },
          {
            name: "email",
          },
          {
            name: "phone",
          },
        ],
        applyNormalization: false,
      },
    ],
    resolutionTechniques: { resolutionType: "ML_MATCHING" },
  };
  try {
```

```
    const command = new CreateMatchingWorkflowCommand(
      createMatchingWorkflowParams,
    );
    const response = await erClient.send(command);

    console.log(
      `Workflow created successfully.\n The workflow ARN is:
 ${response.workflowArn}`,
    );
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Per i dettagli sull'API, CreateMatchingWorkflowconsulta AWS SDK for JavaScript API Reference.

`CreateSchemaMapping`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  CreateSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
```

```
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const createSchemaMappingParamsJson = {
    schemaName: `${data.inputs.schemaNameJson}`,
    mappedInputFields: [
      {
        fieldName: "id",
        type: "UNIQUE_ID",
      },
      {
        fieldName: "name",
        type: "NAME",
      },
      {
        fieldName: "email",
        type: "EMAIL_ADDRESS",
      },
    ],
  };
  const createSchemaMappingParamsCSV = {
    schemaName: `${data.inputs.schemaNameCSV}`,
    mappedInputFields: [
      {
        fieldName: "id",
        type: "UNIQUE_ID",
      },
      {
        fieldName: "name",
        type: "NAME",
      },
      {
        fieldName: "email",
        type: "EMAIL_ADDRESS",
      },
      {
        fieldName: "phone",
        type: "PROVIDER_ID",
        subType: "STRING",
      },
    ],
```

```
  };
  try {
    const command = new CreateSchemaMappingCommand(
      createSchemaMappingParamsJson,
    );
    const response = await erClient.send(command);
    console.log("The JSON schema mapping name is ", response.schemaName);
  } catch (error) {
    console.log("error ", error.message);
  }
};
```

- Per i dettagli sull'API, CreateSchemaMappingconsulta AWS SDK for JavaScript API Reference.

`DeleteMatchingWorkflow`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  DeleteMatchingWorkflowCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });
```

```
export const main = async () => {
  try {
    const deleteWorkflowParams = {
      workflowName: `${data.inputs.workflowName}`,
    };
    const command = new DeleteMatchingWorkflowCommand(deleteWorkflowParams);
    const response = await erClient.send(command);
    console.log("Workflow deleted successfully!", response);
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Per i dettagli sull'API, DeleteMatchingWorkflowconsulta AWS SDK for JavaScript API Reference.

`DeleteSchemaMapping`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  DeleteSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };
```

```
const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const deleteSchemaMapping = {
    schemaName: `${data.inputs.schemaNameJson}`,
  };
  try {
    const command = new DeleteSchemaMappingCommand(deleteSchemaMapping);
    const response = await erClient.send(command);
    console.log("Schema mapping deleted successfully. ", response);
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Per i dettagli sull'API, DeleteSchemaMappingconsulta AWS SDK for JavaScript API Reference.

`GetMatchingJob`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  GetMatchingJobCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
```

```
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  async function getInfo() {
    const getJobInfoParams = {
      workflowName: `${data.inputs.workflowName}`,
      jobId: `${data.inputs.jobId}`,
    };
    try {
      const command = new GetMatchingJobCommand(getJobInfoParams);
      const response = await erClient.send(command);
      console.log(`Job status: ${response.status}`);
    } catch (error) {
      console.log("error ", error.message);
    }
  }
};
```

- Per i dettagli sull'API, GetMatchingJobconsulta AWS SDK for JavaScript API Reference.

`GetSchemaMapping`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

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

```
import {
  GetSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const getSchemaMappingJsonParams = {
    schemaName: `${data.inputs.schemaNameJson}`,
  };
  try {
    const command = new GetSchemaMappingCommand(getSchemaMappingJsonParams);
    const response = await erClient.send(command);
    console.log(response);
    console.log(
      `Schema mapping for the JSON data:\n ${response.mappedInputFields[0]}`,
    );
    console.log("Schema mapping ARN is: ", response.schemaArn);
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Per i dettagli sull'API, GetSchemaMappingconsulta AWS SDK for JavaScript API Reference.

`ListSchemaMappings`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  ListSchemaMappingsCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  async function getInfo() {
    const listSchemaMappingsParams = {
      workflowName: `${data.inputs.workflowName}`,
      jobId: `${data.inputs.jobId}`,
    };
    try {
      const command = new ListSchemaMappingsCommand(listSchemaMappingsParams);
      const response = await erClient.send(command);
      const noOfSchemas = response.schemaList.length;
      for (let i = 0; i < noOfSchemas; i++) {
        console.log(
          `Schema Mapping Name: ${response.schemaList[i].schemaName} `,
        );
      }
    } catch (caught) {
      console.error(caught.message);
      throw caught;
    }
  }
```

- Per i dettagli sull'API, ListSchemaMappingsconsulta AWS SDK for JavaScript API Reference.

`StartMatchingJob`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";
import {
  StartMatchingJobCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const matchingJobOfWorkflowParams = {
    workflowName: `${data.inputs.workflowName}`,
  };
  try {
    const command = new StartMatchingJobCommand(matchingJobOfWorkflowParams);
    const response = await erClient.send(command);
    console.log(`Job ID: ${response.jobID} \n
The matching job was successfully started.`);
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Per i dettagli sull'API, StartMatchingJobconsulta AWS SDK for JavaScript API Reference.

`TagResource`

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

SDK per JavaScript (v3)

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

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  TagResourceCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const tagResourceCommandParams = {
    resourceArn: `${data.inputs.schemaArn}`,
    tags: {
      tag1: "tag1Value",
      tag2: "tag2Value",
    },
  };
  try {
    const command = new TagResourceCommand(tagResourceCommandParams);
    const response = await erClient.send(command);
    console.log("Successfully tagged the resource.");
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Per i dettagli sull'API, TagResourceconsulta AWS SDK for JavaScript API Reference.

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 azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni
- Scenari

Azioni

`PutEvents`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import {
  EventBridgeClient,
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";

export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutEventsCommand({
      Entries: [
        {
          Detail: JSON.stringify({ greeting: "Hello there." }),
          DetailType: detailType,
          Resources: resources,
          Source: source,
        },
      ],
    }),
  );

  console.log("PutEvents response:");
  console.log(response);
  // PutEvents response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- Per i dettagli sull'API, PutEventsconsulta AWS SDK for JavaScript API Reference.

`PutRule`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";

export const putRule = async (
  ruleName = "some-rule",
  source = "some-source",
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutRuleCommand({
      Name: ruleName,
      EventPattern: JSON.stringify({ source: [source] }),
      State: "ENABLED",
      EventBusName: "default",
    }),
  );

  console.log("PutRule response:");
  console.log(response);
  // PutRule response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
```

```
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/
EventBridgeTestRule-1696280037720'
  // }
  return response;
};
```

- Per i dettagli sull'API, PutRuleconsulta AWS SDK for JavaScript API Reference.

`PutTargets`

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

SDK per JavaScript (v3)

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

Importare l'SDK e i moduli client e chiamare l'API.

```
import {
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
```

```
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   FailedEntries: [],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- Per i dettagli sull'API, PutTargetsconsulta AWS SDK for JavaScript API Reference.

Scenari

Utilizzo degli eventi pianificati per invocare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

SDK per JavaScript (v3)

> Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene

richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Esempi di Amazon Glacier con SDK JavaScript for (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Glacier.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`CreateVault`

Il seguente esempio di codice mostra come usare. `CreateVault`

SDK per JavaScript (v3)

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

Crea il client.

```
const { 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 i dettagli sull'API, CreateVaultconsulta AWS SDK for JavaScript API Reference.

`UploadArchive`

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

SDK per JavaScript (v3)

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

Crea il client.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Caricamento dell’archivio.

```
// Load the SDK for JavaScript
import { UploadArchiveCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "./libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME

// Create a new service object and buffer
const buffer = new Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer
const params = { vaultName: vaultname, body: buffer };

const run = async () => {
  try {
    const data = await glacierClient.send(new UploadArchiveCommand(params));
    console.log("Archive ID", data.archiveId);
```

```
    return data; // For unit tests.
  } catch (err) {
    console.log("Error uploading archive!", err);
  }
};
run();
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, UploadArchiveconsulta AWS SDK for JavaScript API Reference.

AWS Glue esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. AWS Glue

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni

Nozioni di base

Salve AWS Glue

L'esempio di codice seguente mostra come iniziare a utilizzare AWS Glue.

SDK per JavaScript (v3)

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

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});

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

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- Per i dettagli sull'API, ListJobsconsulta AWS SDK for JavaScript API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un crawler che esegue la scansione di un bucket Amazon S3 pubblico e genera un database di metadati in formato CSV.
- Elenca le informazioni su database e tabelle nel tuo AWS Glue Data Catalog.
- Crea un processo per estrarre i dati CSV dal bucket S3, trasformare i dati e caricare l'output in formato JSON in un altro bucket S3.
- Elenca le informazioni sulle esecuzioni dei processi, visualizza i dati trasformati e pulisci le risorse.

Per ulteriori informazioni, consulta Tutorial: Guida introduttiva a AWS Glue Studio.

SDK per JavaScript (v3)

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

Creare e avviare un crawler in grado di eseguire il crawling di un bucket pubblico di Amazon Simple Storage Service (Amazon S3) generando un database di metadati che descrive i dati rilevati in formato CSV.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
```

```
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
  } catch {
    return false;
  }
};

/**
 * @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 };
  };
```

Creare e avviare un processo che estrae i dati CSV dal bucket Amazon S3 di origine, li trasforma rimuovendo e rinominando i campi e carica l'output in formato JSON in un altro bucket Amazon S3.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
```

```
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput> }  getJobRun
```

```
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }

  switch (JobRun.JobRunState) {
    case "FAILED":
    case "TIMEOUT":
    case "STOPPED":
    case "ERROR":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "SUCCEEDED":
      return;
    default:
      break;
  }

  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 informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - 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`.

SDK per JavaScript (v3)

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

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
```

```
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

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

- Per i dettagli sull'API, CreateCrawlerconsulta AWS SDK for JavaScript API Reference.

`CreateJob`

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

SDK per JavaScript (v3)

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

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

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

- Per i dettagli sull'API, CreateJobconsulta AWS SDK for JavaScript API Reference.

`DeleteCrawler`

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

SDK per JavaScript (v3)

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

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

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

- Per i dettagli sull'API, DeleteCrawlerconsulta AWS SDK for JavaScript API Reference.

`DeleteDatabase`

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

SDK per JavaScript (v3)

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

```
const deleteDatabase = (databaseName) => {
```

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

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

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

- Per i dettagli sull'API, DeleteDatabaseconsulta AWS SDK for JavaScript API Reference.

`DeleteJob`

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

SDK per JavaScript (v3)

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

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

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

- Per i dettagli sull'API, DeleteJobconsulta AWS SDK for JavaScript API Reference.

`DeleteTable`

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

SDK per JavaScript (v3)

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

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

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

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK for JavaScript API Reference.

`GetCrawler`

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

SDK per JavaScript (v3)

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

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
```

```
  });

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

- Per i dettagli sull'API, GetCrawlerconsulta AWS SDK for JavaScript API Reference.

`GetDatabase`

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

SDK per JavaScript (v3)

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

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

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

- Per i dettagli sull'API, GetDatabaseconsulta AWS SDK for JavaScript API Reference.

`GetDatabases`

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

SDK per JavaScript (v3)

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

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

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

- Per i dettagli sull'API, GetDatabasesconsulta AWS SDK for JavaScript API Reference.

`GetJob`

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

SDK per JavaScript (v3)

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

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

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

- Per i dettagli sull'API, GetJobconsulta AWS SDK for JavaScript API Reference.

`GetJobRun`

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

SDK per JavaScript (v3)

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

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

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

- Per i dettagli sull'API, GetJobRunconsulta AWS SDK for JavaScript API Reference.

`GetJobRuns`

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

SDK per JavaScript (v3)

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

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

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

- Per i dettagli sull'API, GetJobRunsconsulta AWS SDK for JavaScript API Reference.

`GetTables`

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

SDK per JavaScript (v3)

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

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

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

- Per i dettagli sull'API, GetTablesconsulta AWS SDK for JavaScript API Reference.

`ListJobs`

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

SDK per JavaScript (v3)

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

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

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

- Per i dettagli sull'API, ListJobsconsulta AWS SDK for JavaScript API Reference.

`StartCrawler`

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

SDK per JavaScript (v3)

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

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
```

```
};
```

- Per i dettagli sull'API, StartCrawlerconsulta AWS SDK for JavaScript API Reference.

`StartJobRun`

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

SDK per JavaScript (v3)

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

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

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

- Per i dettagli sull'API, StartJobRunconsulta AWS SDK for JavaScript API Reference.

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 azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Salve HealthImaging

L'esempio di codice seguente mostra come iniziare a utilizzare HealthImaging.

SDK per JavaScript (v3)

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListDatastoresReference.

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

Azioni

`CopyImageSet`

Il seguente esempio di codice mostra come usare`CopyImageSet`.

SDK per JavaScript (v3)

Funzione di utilità per copiare un set di immagini.

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 image set.
 * @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: 'xxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       },
    //       sourceImageSetProperties: {
    //             createdAt: 2023-09-22T14:49:26.427Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
    //             latestVersionId: '4',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       }
    // }
    return response;
  } 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(
  "1234567890123456789012345678901 2",
  "1234567890123456789012345678901 2",
  "1",
  "1234567890123456789012345678901 2",
  "1",
  false,
);
```

Copia un sottoinsieme di un set di immagini con una destinazione e forza la copia.

```
await copyImageSet(
  "1234567890123456789012345678901 2",
  "1234567890123456789012345678901 2",
  "1",
  "1234567890123456789012345678901 2",
  "1",
  true,
  ["1234567890123456789012345678901 2", "11223344556677889900112233445566"],
);
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [CopyImageSet](#)Reference.

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

`CreateDatastore`

Il seguente esempio di codice mostra come usare`CreateDatastore`.

SDK per JavaScript (v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //    datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [CreateDatastore](#)Reference.

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

`DeleteDatastore`

Il seguente esempio di codice mostra come usare`DeleteDatastore`.

SDK per JavaScript (v3)

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //       },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [DeleteDatastore](#)Reference.

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

`DeleteImageSet`

Il seguente esempio di codice mostra come usare`DeleteImageSet`.

SDK per JavaScript (v3)

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "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 i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API DeleteImageSetReference.

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

`GetDICOMImportJob`

Il seguente esempio di codice mostra come usare`GetDICOMImportJob`.

SDK per JavaScript (v3)

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
```

```
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta Get DICOMImport Job in AWS SDK for JavaScript API Reference.

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

`GetDatastore`

Il seguente esempio di codice mostra come usare`GetDatastore`.

SDK per JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID }),
  );
  console.log(response);
  // {
```

```
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response.datastoreProperties;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetDatastoreReference.

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

`GetImageFrame`

Il seguente esempio di codice mostra come usare`GetImageFrame`.

SDK per JavaScript (v3)

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 image frame.
```

```
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID",
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    }),
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     contentType: 'application/octet-stream',
  //     imageFrameBlob: <ref *1> IncomingMessage {}
  // }
  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetImageFrameReference.

> **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.

`GetImageSet`

Il seguente esempio di codice mostra come usare`GetImageSet`.

SDK per JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
```

```
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API GetImageSetReference.

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

`GetImageSetMetadata`

Il seguente esempio di codice mostra come usare`GetImageSetMetadata`.

SDK per JavaScript (v3)

Funzione di utilità per ottenere i metadati del set di immagini.

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "node:fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
```

```
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  versionID = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params),
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

Ottiene i metadati del set di immagini senza la versione.

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
```

```
      "12345678901234567890123456789012",
    );
  } catch (err) {
    console.log("Error", err);
  }
```

Ottiene i metadati del set di immagini con la versione.

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1",
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [GetImageSetMetadata](#)Reference.

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

`ListDICOMImportJobs`

Il seguente esempio di codice mostra come usare`ListDICOMImportJobs`.

SDK per JavaScript (v3)

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
```

```
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  const jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page.jobSummaries);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- Per i dettagli sull'API, consulta List DICOMImport Jobs in AWS SDK for JavaScript API Reference.

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

`ListDatastores`

Il seguente esempio di codice mostra come usare`ListDatastores`.

SDK per JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page.datastoreSummaries);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
```

```
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListDatastoresReference.

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

`ListImageSetVersions`

Il seguente esempio di codice mostra come usare`ListImageSetVersions`.

SDK per JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
```

```
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams,
  );

  const imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page.imageSetPropertiesList);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListImageSetVersionsReference.

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

`ListTagsForResource`

Il seguente esempio di codice mostra come usare`ListTagsForResource`.

SDK per JavaScript (v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi",
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }
```

```
  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API ListTagsForResourceReference.

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

`SearchImageSets`

Il seguente esempio di codice mostra come usare`SearchImageSets`.

SDK per JavaScript (v3)

La funzione di utilità per la ricerca di set di immagini.

```
import { paginateSearchImageSets } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters - The
 search criteria filters.
 * @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: operatore EQUAL.

```
  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: operatore BETWEEN che utilizza DICOMStudy data e DICOMStudy ora.

```
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: operatore BETWEEN con createdAt. Gli studi relativi agli orari sono stati precedentemente resi persistenti.

```
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: operatore EQUAL su DICOMSeries instanceUID e BETWEEN su updatedAt e ordina la risposta in ordine ASC nel campo updatedAt.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [
      {
        values: [
          { updatedAt: new Date("1985-04-12T23:20:50.52Z") },
          { updatedAt: new Date() },
        ],
        operator: "BETWEEN",
      },
      {
```

```
          values: [
            {
              DICOMSeriesInstanceUID:
                "1.1.123.123456.1.12.1.1234567890.1234.12345678.123",
            },
          ],
          operator: "EQUAL",
        },
      ],
      sort: {
        sortOrder: "ASC",
        sortField: "updatedAt",
      },
    };

    await searchImageSets(datastoreId, searchCriteria);
  } catch (err) {
    console.error(err);
  }
```

- Per i dettagli sull'API, consulta la sezione API Reference. SearchImageSetsAWS SDK for JavaScript

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

`StartDICOMImportJob`

Il seguente esempio di codice mostra come usare`StartDICOMImportJob`.

SDK per JavaScript (v3)

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
```

```
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
 stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/",
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- Per i dettagli sull'API, consulta Start DICOMImport Job in AWS SDK for JavaScript API Reference.

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

`TagResource`

Il seguente esempio di codice mostra come usare`TagResource`.

SDK per JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                     - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {},
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API TagResourceReference.

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

`UntagResource`

Il seguente esempio di codice mostra come usare`UntagResource`.

SDK per JavaScript (v3)

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
```

```
  //    }
  // }

  return response;
};
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API UntagResourceReference.

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

`UpdateImageSetMetadata`

Il seguente esempio di codice mostra come usare`UpdateImageSetMetadata`.

SDK per JavaScript (v3)

```
import { UpdateImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 * @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: inserisce o aggiorna un attributo e forza 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: rimuove 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 i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API [UpdateImageSetMetadata](#)Reference.

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

Scenari

Nozioni di base su set di immagini e frame di immagini

Il seguente esempio di codice mostra come importare file DICOM e scaricare cornici di immagini. HealthImaging

L'implementazione è strutturata come un'applicazione della riga di comando.

- Impostare le risorse per l'importazione DICOM.
- Importare file DICOM in un datastore.
- Recuperate il set di immagini IDs per il processo di importazione.
- Recuperate la cornice dell'immagine IDs per i set di immagini.
- Scaricare, decodificare e verifica i frame di immagini.
- Eliminare le risorse.

SDK per JavaScript (v3)

Orchestra le fasi (index.js).

```
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 "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
```

```
    name: "Health Imaging Workflow",
    description:
      "Work with DICOM images using an AWS Health Imaging data store.",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-
v|--verbose]",
  });
}
```

Implementa le risorse (deploy-steps.js).

```
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,
  "../../../../../scenarios/features/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 },
);
```

Copia i file DICOM (dataset-steps.js).

```
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.`,
);
```

Avvia l'importazione nel datastore (import-steps.js).

```
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}`,
);
```

Ottieni il set di immagini IDs (image-set-steps.js-).

```
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((ids, next) => {
        return Object.assign({}, ids, {
          [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")}`,
);
```

Ottieni la cornice dell'immagine IDs (image-frame-steps.js).

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node: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;
  },
);
```

Verifica i frame di immagini (verify-steps.js). 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}`,
                  ),
                );
```

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

Elimina le risorse (clean-up-steps.js).

```
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 informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - DeleteImageSet
  - Trova un DICOMImport lavoro
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Avvia DICOMImport 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.

Tagging di un datastore

Il seguente esempio di codice mostra come etichettare un archivio HealthImaging dati.

SDK per JavaScript (v3)

Come taggare un datastore.

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(datastoreArn, tags);
```

```
  } catch (e) {
    console.log(e);
  }
```

La funzione di utilità per taggare 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;
};
```

Come elencare i tag di un datastore.

```
  try {
```

```
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(datastoreArn);
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

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;
};
```

Come rimuovere i tag di un datastore.

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(datastoreArn, keys);
  } catch (e) {
    console.log(e);
  }
```

La funzione di utilità per rimuovere i tag di una risorsa.

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

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

Tagging un set di immagini

Il seguente esempio di codice mostra come etichettare un set di HealthImaging immagini.

SDK per JavaScript (v3)

Come taggare 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 taggare 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;
};
```

Come elencare i tag di 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;
};
```

Come 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 i tag di una risorsa.

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

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

Esempi IAM che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con IAM.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Hello IAM

Il seguente esempio di codice mostra come iniziare a utilizzare IAM.

SDK per JavaScript (v3)

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

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
 async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
  for await (const page of paginator) {
    if (page.Policies) {
      for (const policy of page.Policies) {
        console.log(`${policy.PolicyName}`);
        policyCount++;
      }
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- Per i dettagli sull'API, ListPoliciesconsulta AWS SDK for JavaScript API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come creare un utente e assumere un ruolo.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

- Crea un utente che non disponga di autorizzazioni.
- Crea un ruolo che conceda l'autorizzazione per elencare i bucket Amazon S3 per l'account.
- Aggiungi una policy per consentire all'utente di assumere il ruolo.
- Assumi il ruolo ed elenca i bucket S3 utilizzando le credenziali temporanee, quindi ripulisci le risorse.

SDK per JavaScript (v3)

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

Crea un utente IAM che conceda l'autorizzazione per elencare i bucket Amazon S3. L'utente dispone dei diritti soltanto per assumere il ruolo. Dopo aver assunto il ruolo, utilizza le credenziali temporanee per elencare i bucket per l'account.

```
import {
  CreateUserCommand,
  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 = "iam_basic_test_username";
const policyName = "iam_basic_test_policy";
const roleName = "iam_basic_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;
    }
    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: 120 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

await iamClient.send(
  new DeletePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
  }),
);
```

```
  await iamClient.send(
    new DeleteRoleCommand({
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy

- CreateRole
- CreateUser
- DeleteAccessKey
- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Azioni

`AttachRolePolicy`

Il seguente esempio di codice mostra come usare`AttachRolePolicy`.

SDK per JavaScript (v3)

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

Collega la policy.

```
import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
```

```
    PolicyArn: policyArn,
    RoleName: roleName,
  });

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, AttachRolePolicyconsulta AWS SDK for JavaScript API Reference.

`CreateAccessKey`

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

SDK per JavaScript (v3)

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

Crea la chiave di accesso.

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per i dettagli sull'API, CreateAccessKeyconsulta AWS SDK for JavaScript API Reference.

`CreateAccountAlias`

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

SDK per JavaScript (v3)

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

Crea l'alias dell'account.

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias - A unique name for the account alias.
 * @returns
 */
export const createAccountAlias = (alias) => {
  const command = new CreateAccountAliasCommand({
    AccountAlias: alias,
  });

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateAccountAliasconsulta AWS SDK for JavaScript API Reference.

`CreateGroup`

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

SDK per JavaScript (v3)

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

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

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

- Per i dettagli sull'API, CreateGroupconsulta AWS SDK for JavaScript API Reference.

`CreateInstanceProfile`

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

SDK per JavaScript (v3)

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

```
  const { InstanceProfile } = await iamClient.send(
```

```
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
```

- Per i dettagli sull'API, CreateInstanceProfileconsulta AWS SDK for JavaScript API Reference.

`CreatePolicy`

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

SDK per JavaScript (v3)

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

Crea la policy.

```
import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyName
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: "*",
```

```
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  });

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreatePolicyconsulta AWS SDK for JavaScript API Reference.

`CreateRole`

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

SDK per JavaScript (v3)

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

Crea il ruolo.

```
import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
  const command = new CreateRoleCommand({
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
```

```
          Effect: "Allow",
          Principal: {
            Service: "lambda.amazonaws.com",
          },
          Action: "sts:AssumeRole",
        },
      ],
    }),
    RoleName: roleName,
  });

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

- Per i dettagli sull'API, CreateRoleconsulta AWS SDK for JavaScript API Reference.

`CreateSAMLProvider`

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

SDK per JavaScript (v3)

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

```
import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import * as path from "node: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 i dettagli sull'API, consulta Create SAMLProvider in AWS SDK for JavaScript API Reference.

`CreateServiceLinkedRole`

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

SDK per JavaScript (v3)

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

Crea un ruolo 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" }),
      );
    }
    throw caught;
  }
};
```

- Per i dettagli sull'API, CreateServiceLinkedRoleconsulta AWS SDK for JavaScript API Reference.

`CreateUser`

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

SDK per JavaScript (v3)

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

Creare l'utente.

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const createUser = (name) => {
  const command = new CreateUserCommand({ UserName: name });
  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateUserconsulta AWS SDK for JavaScript API Reference.

`DeleteAccessKey`

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

SDK per JavaScript (v3)

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

Elimina la chiave di accesso.

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const deleteAccessKey = (userName, accessKeyId) => {
  const command = new DeleteAccessKeyCommand({
    AccessKeyId: accessKeyId,
    UserName: userName,
  });

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteAccessKeyconsulta AWS SDK for JavaScript API Reference.

`DeleteAccountAlias`

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

SDK per JavaScript (v3)

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

Elimina l'alias dell'account.

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias
 */
export const deleteAccountAlias = (alias) => {
  const command = new DeleteAccountAliasCommand({ AccountAlias: alias });

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

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteAccountAliasconsulta AWS SDK for JavaScript API Reference.

`DeleteGroup`

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

SDK per JavaScript (v3)

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

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
    GroupName: groupName,
  });

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

- Per i dettagli sull'API, DeleteGroupconsulta AWS SDK for JavaScript API Reference.

`DeleteInstanceProfile`

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

SDK per JavaScript (v3)

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

```
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
```

- Per i dettagli sull'API, DeleteInstanceProfileconsulta AWS SDK for JavaScript API Reference.

`DeletePolicy`

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

SDK per JavaScript (v3)

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

Eliminare il criterio.

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- Per i dettagli sull'API, DeletePolicyconsulta AWS SDK for JavaScript API Reference.

`DeleteRole`

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

SDK per JavaScript (v3)

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

Elimina il ruolo.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Per i dettagli sull'API, DeleteRoleconsulta AWS SDK for JavaScript API Reference.

`DeleteRolePolicy`

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

SDK per JavaScript (v3)

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

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
  });
  return client.send(command);
};
```

- Per i dettagli sull'API, DeleteRolePolicyconsulta AWS SDK for JavaScript API Reference.

`DeleteSAMLProvider`

Il seguente esempio di codice mostra come utilizzare`DeleteSAMLProvider`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
  const command = new DeleteSAMLProviderCommand({
    SAMLProviderArn: providerArn,
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Per i dettagli sull'API, consulta Delete SAMLProvider in AWS SDK for JavaScript API Reference.

`DeleteServerCertificate`

Il seguente esempio di codice mostra come utilizzare`DeleteServerCertificate`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina un certificato del server.

```
import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
  const command = new DeleteServerCertificateCommand({
    ServerCertificateName: certName,
  });

  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteServerCertificateconsulta AWS SDK for JavaScript API Reference.

`DeleteServiceLinkedRole`

Il seguente esempio di codice mostra come utilizzare`DeleteServiceLinkedRole`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Per i dettagli sull'API, DeleteServiceLinkedRoleconsulta AWS SDK for JavaScript API Reference.

`DeleteUser`

Il seguente esempio di codice mostra come utilizzare`DeleteUser`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare l'utente.

```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
  const command = new DeleteUserCommand({ UserName: name });
  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteUserconsulta AWS SDK for JavaScript API Reference.

`DetachRolePolicy`

Il seguente esempio di codice mostra come utilizzare`DetachRolePolicy`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Scollega la policy.

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const detachRolePolicy = (policyArn, roleName) => {
  const command = new DetachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DetachRolePolicyconsulta AWS SDK for JavaScript API Reference.

`GetAccessKeyLastUsed`

Il seguente esempio di codice mostra come utilizzare`GetAccessKeyLastUsed`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la chiave di accesso.

```
import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
 *
 * @param {string} accessKeyId
 */
export const getAccessKeyLastUsed = async (accessKeyId) => {
  const command = new GetAccessKeyLastUsedCommand({
    AccessKeyId: accessKeyId,
  });

  const response = await client.send(command);

  if (response.AccessKeyLastUsed?.LastUsedDate) {
    console.log(`
    ${accessKeyId} was last used by ${response.UserName} via
    the ${response.AccessKeyLastUsed.ServiceName} service on
    ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
    `);
  }

  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, GetAccessKeyLastUsedconsulta AWS SDK for JavaScript API Reference.

`GetAccountPasswordPolicy`

Il seguente esempio di codice mostra come utilizzare`GetAccountPasswordPolicy`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la policy sulla password dell’account.

```
import {
```

```
  GetAccountPasswordPolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const getAccountPasswordPolicy = async () => {
  const command = new GetAccountPasswordPolicyCommand({});

  const response = await client.send(command);
  console.log(response.PasswordPolicy);
  return response;
};
```

- Per i dettagli sull'API, GetAccountPasswordPolicyconsulta AWS SDK for JavaScript API Reference.

`GetPolicy`

Il seguente esempio di codice mostra come utilizzare`GetPolicy`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la policy.

```
import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const getPolicy = (policyArn) => {
```

```
  const command = new GetPolicyCommand({
    PolicyArn: policyArn,
  });

  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, GetPolicyconsulta AWS SDK for JavaScript API Reference.

`GetRole`

Il seguente esempio di codice mostra come utilizzare`GetRole`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera il ruolo.

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const getRole = (roleName) => {
  const command = new GetRoleCommand({
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, GetRoleconsulta AWS SDK for JavaScript API Reference.

`GetServerCertificate`

Il seguente esempio di codice mostra come utilizzare`GetServerCertificate`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera un certificato del server.

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, GetServerCertificateconsulta AWS SDK for JavaScript API Reference.

`GetServiceLinkedRoleDeletionStatus`

Il seguente esempio di codice mostra come utilizzare`GetServiceLinkedRoleDeletionStatus`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, GetServiceLinkedRoleDeletionStatusconsulta AWS SDK for JavaScript API Reference.

`ListAccessKeys`

Il seguente esempio di codice mostra come utilizzare`ListAccessKeys`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le chiavi di accesso.

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
  const command = new ListAccessKeysCommand({
    MaxItems: 5,
    UserName: userName,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.AccessKeyMetadata?.length) {
    for (const key of response.AccessKeyMetadata) {
      yield key;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccessKeysCommand({
          Marker: response.Marker,
        }),
```

```
      );
    } else {
      break;
    }
  }
}
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListAccessKeysconsulta AWS SDK for JavaScript API Reference.

`ListAccountAliases`

Il seguente esempio di codice mostra come utilizzare`ListAccountAliases`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca gli alias di un account.

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 */
export async function* listAccountAliases() {
  const command = new ListAccountAliasesCommand({ MaxItems: 5 });

  let response = await client.send(command);

  while (response.AccountAliases?.length) {
```

```
    for (const alias of response.AccountAliases) {
      yield alias;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccountAliasesCommand({
          Marker: response.Marker,
          MaxItems: 5,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListAccountAliasesconsulta AWS SDK for JavaScript API Reference.

`ListAttachedRolePolicies`

Il seguente esempio di codice mostra come utilizzare`ListAttachedRolePolicies`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le policy collegate a un ruolo.

```
import {
  ListAttachedRolePoliciesCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
  const command = new ListAttachedRolePoliciesCommand({
    RoleName: roleName,
  });

  let response = await client.send(command);

  while (response.AttachedPolicies?.length) {
    for (const policy of response.AttachedPolicies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAttachedRolePoliciesCommand({
          RoleName: roleName,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Per i dettagli sull'API, ListAttachedRolePoliciesconsulta AWS SDK for JavaScript API Reference.

`ListGroups`

Il seguente esempio di codice mostra come utilizzare`ListGroups`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i gruppi.

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 */
export async function* listGroups() {
  const command = new ListGroupsCommand({
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.Groups?.length) {
    for (const group of response.Groups) {
      yield group;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListGroupsCommand({
          Marker: response.Marker,
          MaxItems: 10,
        }),
      );
    } else {
      break;
    }
  }
```

```
}
```

- Per i dettagli sull'API, ListGroupsconsulta AWS SDK for JavaScript API Reference.

`ListPolicies`

Il seguente esempio di codice mostra come utilizzare`ListPolicies`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le policy.

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listPolicies() {
  const command = new ListPoliciesCommand({
    MaxItems: 10,
    OnlyAttached: false,
    // List only the customer managed policies in your Amazon Web Services account.
    Scope: "Local",
  });

  let response = await client.send(command);

  while (response.Policies?.length) {
    for (const policy of response.Policies) {
```

```
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListPoliciesCommand({
          Marker: response.Marker,
          MaxItems: 10,
          OnlyAttached: false,
          Scope: "Local",
        }),
      );
    } else {
      break;
    }
  }
}
```

- Per i dettagli sull'API, ListPoliciesconsulta AWS SDK for JavaScript API Reference.

`ListRolePolicies`

Il seguente esempio di codice mostra come utilizzare`ListRolePolicies`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le policy.

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
```

```
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 * @param {string} roleName
 */
export async function* listRolePolicies(roleName) {
  const command = new ListRolePoliciesCommand({
    RoleName: roleName,
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.PolicyNames?.length) {
    for (const policyName of response.PolicyNames) {
      yield policyName;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolePoliciesCommand({
          RoleName: roleName,
          MaxItems: 10,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Per i dettagli sull'API, ListRolePoliciesconsulta AWS SDK for JavaScript API Reference.

`ListRoles`

Il seguente esempio di codice mostra come utilizzare`ListRoles`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i ruoli.

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listRoles() {
  const command = new ListRolesCommand({
    MaxItems: 10,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.Roles?.length) {
    for (const role of response.Roles) {
      yield role;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolesCommand({
          Marker: response.Marker,
        }),
      );
    } else {
```

```
      break;
    }
  }
}
```

- Per i dettagli sull'API, ListRolesconsulta AWS SDK for JavaScript API Reference.

`ListSAMLProviders`

Il seguente esempio di codice mostra come utilizzare`ListSAMLProviders`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i provider SAML.

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listSamlProviders = async () => {
  const command = new ListSAMLProvidersCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Per i dettagli sull'API, consulta List SAMLProviders in AWS SDK for JavaScript API Reference.

`ListServerCertificates`

Il seguente esempio di codice mostra come utilizzare`ListServerCertificates`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i certificati.

```
import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listServerCertificates() {
  const command = new ListServerCertificatesCommand({});
  let response = await client.send(command);

  while (response.ServerCertificateMetadataList?.length) {
    for await (const cert of response.ServerCertificateMetadataList) {
      yield cert;
    }

    if (response.IsTruncated) {
      response = await client.send(new ListServerCertificatesCommand({}));
    } else {
      break;
    }
  }
}
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListServerCertificatesconsulta AWS SDK for JavaScript API Reference.

`ListUsers`

Il seguente esempio di codice mostra come utilizzare`ListUsers`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca gli utenti.

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listUsers = async () => {
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);

  for (const { UserName, CreateDate } of response.Users) {
    console.log(`${UserName} created on: ${CreateDate}`);
  }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListUsersconsulta AWS SDK for JavaScript API Reference.

`PutRolePolicy`

Il seguente esempio di codice mostra come utilizzare`PutRolePolicy`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::amzn-s3-demo-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Per i dettagli sull'API, PutRolePolicyconsulta AWS SDK for JavaScript API Reference.

`UpdateAccessKey`

Il seguente esempio di codice mostra come utilizzare`UpdateAccessKey`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna la chiave di accesso.

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const updateAccessKey = (userName, accessKeyId) => {
  const command = new UpdateAccessKeyCommand({
    AccessKeyId: accessKeyId,
    Status: StatusType.Inactive,
    UserName: userName,
  });

  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, UpdateAccessKeyconsulta AWS SDK for JavaScript API Reference.

`UpdateServerCertificate`

Il seguente esempio di codice mostra come utilizzare`UpdateServerCertificate`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna un certificato del server.

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
```

```
 * @param {string} currentName
 * @param {string} newName
 */
export const updateServerCertificate = (currentName, newName) => {
  const command = new UpdateServerCertificateCommand({
    ServerCertificateName: currentName,
    NewServerCertificateName: newName,
  });

  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, UpdateServerCertificateconsulta AWS SDK for JavaScript API Reference.

`UpdateUser`

Il seguente esempio di codice mostra come utilizzare`UpdateUser`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna l'utente.

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentUserName
 * @param {string} newUserName
 */
export const updateUser = (currentUserName, newUserName) => {
  const command = new UpdateUserCommand({
```

```
    UserName: currentUserName,
    NewUserName: newUserName,
  });

  return client.send(command);
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, UpdateUserconsulta AWS SDK for JavaScript API Reference.

`UploadServerCertificate`

Il seguente esempio di codice mostra come utilizzare`UploadServerCertificate`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "node:path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
 the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
```

```
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }

    throw err;
  }
};

/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, UploadServerCertificateconsulta AWS SDK for JavaScript API Reference.

Scenari

Creazione e gestione di un servizio resiliente

L'esempio di codice seguente 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 (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze in un intervallo specificato.
- Gestisci e distribuisci le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo Auto Scaling 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.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui lo scenario interattivo al prompt dei comandi.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
```

```
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-
v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}
```

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),
        );
      }
      return MESSAGES.noIpRules;
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      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("${IP_ADDRESS}", state.myIp);
    }
    return false;
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${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(
```

```
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${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,
      );
    }
    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);
  }
  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,
    );
  }
  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);
  }
  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,
      );
    }
    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,
      );
    }
    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(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }
  return MESSAGES.deletedAutoScalingGroup.replace(
    "${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(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
  return MESSAGES.deletedLaunchTemplate.replace(
    "${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(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${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(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${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);
    }
    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);
  }
  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");
  }
  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(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${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,
      );
```

```
    }
    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,
      );
    }
    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(
        "${IP}",
        state.myIp,
      );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${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;
    }
    console.log(err.name);
    throw err;
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
```

```
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup

- UpdateAutoScalingGroup

AWS IoT SiteWise 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 IoT SiteWise

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni

Nozioni di base

Salve AWS IoT SiteWise

L'esempio di codice seguente mostra come iniziare a utilizzare AWS IoT SiteWise.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  paginateListAssetModels,
```

```
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";

// Call ListDocuments and display the result.
export const main = async () => {
  const client = new IoTSiteWiseClient();
  const listAssetModelsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListAssetModels({ client }, { maxResults: 5 });
    for await (const page of paginator) {
      listAssetModelsPaginated.push(...page.assetModelSummaries);
    }
  } catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
  }
  for (const { name, creationDate } of listAssetModelsPaginated) {
    console.log(`${name} - ${creationDate}`);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Per i dettagli sull'API, ListAssetModelsconsulta AWS SDK for JavaScript API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un modello di AWS IoT SiteWise asset.
- Crea una AWS IoT SiteWise risorsa.

- Recuperare i valori dell'ID della proprietà.
- Invia dati a una AWS IoT SiteWise risorsa.
- Recupera il valore della proprietà AWS IoT SiteWise Asset.
- Crea un AWS IoT SiteWise portale.
- Crea un AWS IoT SiteWise gateway.
- Descrivi il AWS IoT SiteWise Gateway.
- Eliminare gli AWS IoT SiteWise asset.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
  //} from "@aws-doc-sdk-examples/lib/scenario/index.js";
} from "../../libs/scenario/index.js";
import {
  IoTSiteWiseClient,
  CreateAssetModelCommand,
  CreateAssetCommand,
  ListAssetModelPropertiesCommand,
  BatchPutAssetPropertyValueCommand,
  GetAssetPropertyValueCommand,
  CreatePortalCommand,
  DescribePortalCommand,
  CreateGatewayCommand,
  DescribeGatewayCommand,
  DeletePortalCommand,
  DeleteGatewayCommand,
  DeleteAssetCommand,
  DeleteAssetModelCommand,
  DescribeAssetModelCommand,
```

```
} from "@aws-sdk/client-iotsitewise";
import {
  CloudFormationClient,
  CreateStackCommand,
  DeleteStackCommand,
  DescribeStacksCommand,
  waitUntilStackExists,
  waitUntilStackCreateComplete,
  waitUntilStackDeleteComplete,
} from "@aws-sdk/client-cloudformation";
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { parseArgs } from "node:util";
import { readFileSync } from "node:fs";
import { fileURLToPath } from "node:url";
import { dirname } from "node:path";

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);
const stackName = "SiteWiseBasicsStack";

/**
 * @typedef {{
 *   iotSiteWiseClient: import('@aws-sdk/client-iotsitewise').IotSiteWiseClient,
 *   cloudFormationClient: import('@aws-sdk/client-
cloudformation').CloudFormationClient,
 *   stackName,
 *   stack,
 *   askToDeleteResources: true,
 *   asset: {assetName: "MyAsset1"},
 *   assetModel: {assetModelName: "MyAssetModel1"},
 *   portal: {portalName: "MyPortal1"},
 *   gateway: {gatewayName: "MyGateway1"},
 *   propertyIds: [],
 *   contactEmail: "user@mydomain.com",
 *   thing: "MyThing1",
 *   sampleData: { temperature: 23.5, humidity: 65.0}
 * }} State
 */

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
```

```
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `AWS IoT SiteWise is a fully managed industrial software-as-a-service (SaaS)
 that makes it easy to collect, store, organize, and monitor data from industrial
 equipment and processes. It is designed to help industrial and manufacturing
 organizations collect data from their equipment and processes, and use that data to
 make informed decisions about their operations.
One of the key features of AWS IoT SiteWise is its ability to connect to a wide
 range of industrial equipment and systems, including programmable logic controllers
 (PLCs), sensors, and other industrial devices. It can collect data from these
 devices and organize it into a unified data model, making it easier to analyze and
 gain insights from the data. AWS IoT SiteWise also provides tools for visualizing
 the data, setting up alarms and alerts, and generating reports.
Another key feature of AWS IoT SiteWise is its ability to scale to handle large
 volumes of data. It can collect and store data from thousands of devices and
 process millions of data points per second, making it suitable for large-scale
 industrial operations. Additionally, AWS IoT SiteWise is designed to be secure
 and compliant, with features like role-based access controls, data encryption,
 and integration with other AWS services for additional security and compliance
 features.

Let's get started...`,
  { header: true },
);

const displayBuildCloudFormationStack = new ScenarioOutput(
  "displayBuildCloudFormationStack",
  "This scenario uses AWS CloudFormation to create an IAM role that is required for
 this scenario. The stack will now be deployed.",
);

const sdkBuildCloudFormationStack = new ScenarioAction(
  "sdkBuildCloudFormationStack",
  async (/** @type {State} */ state) => {
    try {
      const data = readFileSync(
        `${__dirname}/../../../../resources/cfn/iotsitewise_basics/SitewiseRoles-
template.yml`,
        "utf8",
      );
      await state.cloudFormationClient.send(
```

```
        new CreateStackCommand({
          StackName: stackName,
          TemplateBody: data,
          Capabilities: ["CAPABILITY_IAM"],
        }),
      );
      await waitUntilStackExists(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      await waitUntilStackCreateComplete(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      const stack = await state.cloudFormationClient.send(
        new DescribeStacksCommand({
          StackName: stackName,
        }),
      );
      state.stack = stack.Stacks[0].Outputs[0];
      console.log(`The ARN of the IAM role is ${state.stack.OutputValue}`);
    } catch (caught) {
      console.error(caught.message);
      throw caught;
    }
  },
);

const displayCreateAWSSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSSiteWiseAssetModel",
  `1. Create an AWS SiteWise Asset Model
An AWS IoT SiteWise Asset Model is a way to represent the physical assets, such
 as equipment, processes, and systems, that exist in an industrial environment.
 This model provides a structured and hierarchical representation of these assets,
 allowing users to define the relationships and properties of each asset.

This scenario creates two asset model properties: temperature and humidity.`,
);

const sdkCreateAWSSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSSiteWiseAssetModel",
  async (/** @type {State} */ state) => {
    let assetModelResponse;
    try {
```

```
      assetModelResponse = await state.iotSiteWiseClient.send(
        new CreateAssetModelCommand({
          assetModelName: state.assetModel.assetModelName,
          assetModelProperties: [
            {
              name: "Temperature",
              dataType: "DOUBLE",
              type: {
                measurement: {},
              },
            },
            {
              name: "Humidity",
              dataType: "DOUBLE",
              type: {
                measurement: {},
              },
            },
          ],
        }),
      );
      state.assetModel.assetModelId = assetModelResponse.assetModelId;
      console.log(
        `Asset Model successfully created. Asset Model ID:
 ${state.assetModel.assetModelId}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceAlreadyExistsException") {
        console.log(
          `The Asset Model ${state.assetModel.assetModelName} already exists.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayCreateAWSIoTSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSIoTSiteWiseAssetModel",
  `2. Create an AWS IoT SiteWise Asset
The IoT SiteWise model that we just created defines the structure and metadata for
 your physical assets. Now we create an asset from the asset model.
```

```
Let's wait 30 seconds for the asset to be ready.`,
);

const waitThirtySeconds = new ScenarioAction("waitThirtySeconds", async () => {
  await wait(30); // wait 30 seconds
  console.log("Time's up! Let's check the asset's status.");
});

const sdkCreateAWSIoTSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSIoTSiteWiseAssetModel",
  async (/** @type {State} */ state) => {
    try {
      const assetResponse = await state.iotSiteWiseClient.send(
        new CreateAssetCommand({
          assetModelId: state.assetModel.assetModelId,
          assetName: state.asset.assetName,
        }),
      );
      state.asset.assetId = assetResponse.assetId;
      console.log(`Asset created with ID: ${state.asset.assetId}`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(
          `The Asset ${state.assetModel.assetModelName} was not found.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayRetrievePropertyId = new ScenarioOutput(
  "displayRetrievePropertyId",
  `3. Retrieve the property ID values

To send data to an asset, we need to get the property ID values. In this scenario,
 we access the temperature and humidity property ID values.`,
);

const sdkRetrievePropertyId = new ScenarioAction(
  "sdkRetrievePropertyId",
```

```
  async (state) => {
    try {
      const retrieveResponse = await state.iotSiteWiseClient.send(
        new ListAssetModelPropertiesCommand({
          assetModelId: state.assetModel.assetModelId,
        }),
      );
      for (const retrieveResponseKey in
 retrieveResponse.assetModelPropertySummaries) {
        if (
          retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
            .name === "Humidity"
        ) {
          state.propertyIds.Humidity =
            retrieveResponse.assetModelPropertySummaries[
              retrieveResponseKey
            ].id;
        }
        if (
          retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
            .name === "Temperature"
        ) {
          state.propertyIds.Temperature =
            retrieveResponse.assetModelPropertySummaries[
              retrieveResponseKey
            ].id;
        }
      }
      console.log(`The Humidity propertyId is ${state.propertyIds.Humidity}`);
      console.log(
        `The Temperature propertyId is ${state.propertyIds.Temperature}`,
      );
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem retrieving the properties: ${caught.message}`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);
```

```
const displaySendDataToIoTSiteWiseAsset = new ScenarioOutput(
  "displaySendDataToIoTSiteWiseAsset",
  `4. Send data to an AWS IoT SiteWise Asset

By sending data to an IoT SiteWise Asset, you can aggregate data from multiple
 sources, normalize the data into a standard format, and store it in a centralized
 location. This makes it easier to analyze and gain insights from the data.

In this example, we generate sample temperature and humidity data and send it to the
 AWS IoT SiteWise asset.`,
);

const sdkSendDataToIoTSiteWiseAsset = new ScenarioAction(
  "sdkSendDataToIoTSiteWiseAsset",
  async (state) => {
    try {
      const sendResponse = await state.iotSiteWiseClient.send(
        new BatchPutAssetPropertyValueCommand({
          entries: [
            {
              entryId: "entry-3",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Humidity,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.humidity,
                  },
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
            {
              entryId: "entry-4",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Temperature,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.temperature,
                  },
```

```
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
          ],
        }),
      );
      console.log("The data was sent successfully.");
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayRetrieveValueOfIoTSiteWiseAsset = new ScenarioOutput(
  "displayRetrieveValueOfIoTSiteWiseAsset",
  `5. Retrieve the value of the IoT SiteWise Asset property

IoT SiteWise is an AWS service that allows you to collect, process, and analyze
 industrial data from connected equipment and sensors. One of the key benefits of
 reading an IoT SiteWise property is the ability to gain valuable insights from your
 industrial data.`,
);

const sdkRetrieveValueOfIoTSiteWiseAsset = new ScenarioAction(
  "sdkRetrieveValueOfIoTSiteWiseAsset",
  async (/** @type {State} */ state) => {
    try {
      const temperatureResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
          propertyId: state.propertyIds.Temperature,
        }),
      );
      const humidityResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
```

```
          propertyId: state.propertyIds.Humidity,
        }),
      );
      console.log(
        `The property value for Temperature is
 ${temperatureResponse.propertyValue.value.doubleValue}`,
      );
      console.log(
        `The property value for Humidity is
 ${humidityResponse.propertyValue.value.doubleValue}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayCreateIoTSiteWisePortal = new ScenarioOutput(
  "displayCreateIoTSiteWisePortal",
  `6. Create an IoT SiteWise Portal

An IoT SiteWise Portal allows you to aggregate data from multiple industrial
 sources, such as sensors, equipment, and control systems, into a centralized
 platform.`,
);

const sdkCreateIoTSiteWisePortal = new ScenarioAction(
  "sdkCreateIoTSiteWisePortal",
  async (/** @type {State} */ state) => {
    try {
      const createPortalResponse = await state.iotSiteWiseClient.send(
        new CreatePortalCommand({
          portalName: state.portal.portalName,
          portalContactEmail: state.contactEmail,
          roleArn: state.stack.OutputValue,
        }),
      );
      state.portal = { ...state.portal, ...createPortalResponse };
      await wait(5); // Allow the portal to properly propagate.
```

```
      console.log(
        `Portal created successfully. Portal ID ${createPortalResponse.portalId}`,
      );
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem creating the Portal: ${caught.message}.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayDescribePortal = new ScenarioOutput(
  "displayDescribePortal",
  `7. Describe the Portal

In this step, we get a description of the portal and display the portal URL.`,
);

const sdkDescribePortal = new ScenarioAction(
  "sdkDescribePortal",
  async (/** @type {State} */ state) => {
    try {
      const describePortalResponse = await state.iotSiteWiseClient.send(
        new DescribePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(`Portal URL: ${describePortalResponse.portalStartUrl}`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);
```

```
const displayCreateIoTSiteWiseGateway = new ScenarioOutput(
  "displayCreateIoTSiteWiseGateway",
  `8. Create an IoT SiteWise Gateway

IoT SiteWise Gateway serves as the bridge between industrial equipment, sensors, and
 the cloud-based IoT SiteWise service. It is responsible for securely collecting,
 processing, and transmitting data from various industrial assets to the IoT
 SiteWise platform, enabling real-time monitoring, analysis, and optimization of
 industrial operations.`,
);

const sdkCreateIoTSiteWiseGateway = new ScenarioAction(
  "sdkCreateIoTSiteWiseGateway",
  async (/** @type {State} */ state) => {
    try {
      const createGatewayResponse = await state.iotSiteWiseClient.send(
        new CreateGatewayCommand({
          gatewayName: state.gateway.gatewayName,
          gatewayPlatform: {
            greengrassV2: {
              coreDeviceThingName: state.thing,
            },
          },
        }),
      );
      console.log(
        `Gateway creation completed successfully. ID is
 ${createGatewayResponse.gatewayId}`,
      );
      state.gateway.gatewayId = createGatewayResponse.gatewayId;
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem creating the gateway: ${caught.message}.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayDescribeIoTSiteWiseGateway = new ScenarioOutput(
```

```
  "displayDescribeIoTSiteWiseGateway",
  "9. Describe the IoT SiteWise Gateway",
);

const sdkDescribeIoTSiteWiseGateway = new ScenarioAction(
  "sdkDescribeIoTSiteWiseGateway",
  async (/** @type {State} */ state) => {
    try {
      const describeGatewayResponse = await state.iotSiteWiseClient.send(
        new DescribeGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log("Gateway creation completed successfully.");
      console.log(`Gateway Name: ${describeGatewayResponse.gatewayName}`);
      console.log(`Gateway ARN: ${describeGatewayResponse.gatewayArn}`);
      console.log(
        `Gateway Platform: ${Object.keys(describeGatewayResponse.gatewayPlatform)}`,
      );
      console.log(
        `Gateway Creation Date: ${describeGatewayResponse.creationDate}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const askToDeleteResources = new ScenarioInput(
  "askToDeleteResources",
  `10. Delete the AWS IoT SiteWise Assets

Before you can delete the Asset Model, you must delete the assets.`,
  { type: "confirm" },
);

const displayConfirmDeleteResources = new ScenarioAction(
  "displayConfirmDeleteResources",
  async (/** @type {State} */ state) => {
```

```
    if (state.askToDeleteResources) {
      return "You selected to delete the SiteWise assets.";
    }
    return "The resources will not be deleted. Please delete them manually to avoid
 charges.";
  },
);

const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    await wait(10); // Give the portal status time to catch up.
    try {
      await state.iotSiteWiseClient.send(
        new DeletePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(
        `Portal ${state.portal.portalName} was deleted successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
      } else {
        console.log(`When trying to delete the portal: ${caught.message}`);
      }
    }

    try {
      await state.iotSiteWiseClient.send(
        new DeleteGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log(
        `Gateway ${state.gateway.gatewayName} was deleted successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
      } else {
        console.log(`When trying to delete the gateway: ${caught.message}`);
      }
```

```
  }

  try {
    await state.iotSiteWiseClient.send(
      new DeleteAssetCommand({
        assetId: state.asset.assetId,
      }),
    );
    await wait(5); // Allow the delete to finish.
    console.log(`Asset ${state.asset.assetName} was deleted successfully.`);
  } catch (caught) {
    if (caught.name === "ResourceNotFoundException") {
      console.log(`The Asset ${state.asset.assetName} was not found.`);
    } else {
      console.log(`When deleting the asset: ${caught.message}`);
    }
  }

  await wait(30); // Allow asset deletion to finish.
  try {
    await state.iotSiteWiseClient.send(
      new DeleteAssetModelCommand({
        assetModelId: state.assetModel.assetModelId,
      }),
    );
    console.log(
      `Asset Model ${state.assetModel.assetModelName} was deleted successfully.`,
    );
  } catch (caught) {
    if (caught.name === "ResourceNotFoundException") {
      console.log(
        `The Asset Model ${state.assetModel.assetModelName} was not found.`,
      );
    } else {
      console.log(`When deleting the asset model: ${caught.message}`);
    }
  }

  try {
    await state.cloudFormationClient.send(
      new DeleteStackCommand({
        StackName: stackName,
      }),
    );
```

```
      await waitUntilStackDeleteComplete(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      console.log("The stack was deleted successfully.");
    } catch (caught) {
      console.log(
        `${caught.message}. The stack was NOT deleted. Please clean up the resources
 manually.`,
      );
    }
  },
  { skipWhen: (/** @type {{}} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the IoT Sitewise Basics scenario for the AWS Javascript SDK v3.
 Thank you!",
);

const myScenario = new Scenario(
  "IoTSiteWise Basics",
  [
    greet,
    pressEnter,
    displayBuildCloudFormationStack,
    sdkBuildCloudFormationStack,
    pressEnter,
    displayCreateAWSSiteWiseAssetModel,
    sdkCreateAWSSiteWiseAssetModel,
    displayCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    waitThirtySeconds,
    sdkCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    displayRetrievePropertyId,
    sdkRetrievePropertyId,
    pressEnter,
    displaySendDataToIoTSiteWiseAsset,
    sdkSendDataToIoTSiteWiseAsset,
    pressEnter,
    displayRetrieveValueOfIoTSiteWiseAsset,
    sdkRetrieveValueOfIoTSiteWiseAsset,
```

```
    pressEnter,
    displayCreateIoTSiteWisePortal,
    sdkCreateIoTSiteWisePortal,
    pressEnter,
    displayDescribePortal,
    sdkDescribePortal,
    pressEnter,
    displayCreateIoTSiteWiseGateway,
    sdkCreateIoTSiteWiseGateway,
    pressEnter,
    displayDescribeIoTSiteWiseGateway,
    sdkDescribeIoTSiteWiseGateway,
    pressEnter,
    askToDeleteResources,
    displayConfirmDeleteResources,
    sdkDeleteResources,
    goodbye,
  ],
  {
    iotSiteWiseClient: new IoTSiteWiseClient({}),
    cloudFormationClient: new CloudFormationClient({}),
    asset: { assetName: "MyAsset1" },
    assetModel: { assetModelName: "MyAssetModel1" },
    portal: { portalName: "MyPortal1" },
    gateway: { gatewayName: "MyGateway1" },
    propertyIds: [],
    contactEmail: "user@mydomain.com",
    thing: "MyThing1",
    sampleData: { temperature: 23.5, humidity: 65.0 },
  },
);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
```

```
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

Azioni

`BatchPutAssetPropertyValue`

Il seguente esempio di codice mostra come usare`BatchPutAssetPropertyValue`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  BatchPutAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Batch put asset property values.
 * @param {{ entries : array }}
 */
export const main = async ({ entries }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new BatchPutAssetPropertyValueCommand({
        entries: entries,
      }),
    );
    console.log("Asset properties batch put successfully.");
```

```
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(`${caught.message}. A resource could not be found.`);
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, BatchPutAssetPropertyValueconsulta AWS SDK for JavaScript API Reference.

`CreateAsset`

Il seguente esempio di codice mostra come utilizzare`CreateAsset`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  CreateAssetCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an Asset.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetName, assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateAssetCommand({
```

```
        assetName: assetName, // The name to give the Asset.
        assetModelId: assetModelId, // The ID of the asset model from which to
 create the asset.
      }),
    );
    console.log("Asset created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset model could not be found. Please check the
 asset model id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, CreateAssetconsulta AWS SDK for JavaScript API Reference.

`CreateAssetModel`

Il seguente esempio di codice mostra come utilizzare`CreateAssetModel`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  CreateAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an Asset Model.
```

```
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetModelName, assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateAssetModelCommand({
        assetModelName: assetModelName, // The name to give the Asset Model.
      }),
    );
    console.log("Asset model created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the asset model.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, CreateAssetModelconsulta AWS SDK for JavaScript API Reference.

`CreateGateway`

Il seguente esempio di codice mostra come utilizzare`CreateGateway`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  CreateGatewayCommand,
  IoTSiteWiseClient,
```

```
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create a Gateway.
 * @param {{  }}
 */
export const main = async ({ gatewayName }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateGatewayCommand({
        gatewayName: gatewayName, // The name to give the created Gateway.
      }),
    );
    console.log("Gateway created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the Gateway.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, CreateGatewayconsulta AWS SDK for JavaScript API Reference.

`DeleteAsset`

Il seguente esempio di codice mostra come utilizzare`DeleteAsset`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  DeleteAssetCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset.
 * @param {{ assetId : string }}
 */
export const main = async ({ assetId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetCommand({
        assetId: assetId, // The model id to delete.
      }),
    );
    console.log("Asset deleted successfully.");
    return { assetDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, DeleteAssetconsulta AWS SDK for JavaScript API Reference.

`DeleteAssetModel`

Il seguente esempio di codice mostra come utilizzare`DeleteAssetModel`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  DeleteAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetModelCommand({
        assetModelId: assetModelId, // The model id to delete.
      }),
    );
    console.log("Asset model deleted successfully.");
    return { assetModelDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset model.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, DeleteAssetModelconsulta AWS SDK for JavaScript API Reference.

`DeleteGateway`

Il seguente esempio di codice mostra come utilizzare`DeleteGateway`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  DeleteGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway deleted successfully.");
    return { gatewayDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Gateway could not be found. Please check the Gateway
 Id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, DeleteGatewayconsulta AWS SDK for JavaScript API Reference.

`DescribeAssetModel`

Il seguente esempio di codice mostra come utilizzare`DescribeAssetModel`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  DescribeAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const { assetModelDescription } = await client.send(
      new DescribeAssetModelCommand({
        assetModelId: assetModelId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Asset model information retrieved successfully.");
    return { assetModelDescription: assetModelDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset model could not be found. Please check the
 asset model id.`,
```

```
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, DescribeAssetModelconsulta AWS SDK for JavaScript API Reference.

`DescribeGateway`

Il seguente esempio di codice mostra come utilizzare`DescribeGateway`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  DescribeGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const { gatewayDescription } = await client.send(
      new DescribeGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway information retrieved successfully.");
```

```
    return { gatewayDescription: gatewayDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Gateway could not be found. Please check the Gateway
 Id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, DescribeGatewayconsulta AWS SDK for JavaScript API Reference.

`GetAssetPropertyValue`

Il seguente esempio di codice mostra come utilizzare`GetAssetPropertyValue`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  GetAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset property value.
 * @param {{ entryId : string }}
 */
export const main = async ({ entryId }) => {
  const client = new IoTSiteWiseClient({});
  try {
```

```
    const result = await client.send(
      new GetAssetPropertyValueCommand({
        entryId: entryId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Asset property information retrieved successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset property entry could not be found. Please
 check the entry id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, GetAssetPropertyValueconsulta AWS SDK for JavaScript API Reference.

`ListAssetModels`

Il seguente esempio di codice mostra come utilizzare`ListAssetModels`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  ListAssetModelsCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
```

```
 * List asset models.
 * @param {{ assetModelTypes : array }}
 */
export const main = async ({ assetModelTypes = [] }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new ListAssetModelsCommand({
        assetModelTypes: assetModelTypes, // The model types to list
      }),
    );
    console.log("Asset model types retrieved successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem listing the asset model types.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, ListAssetModelsconsulta AWS SDK for JavaScript API Reference.

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.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

- Esempi serverless

Azioni

`PutRecords`

Il seguente esempio di codice mostra come utilizzare. `PutRecords`

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { PutRecordsCommand, KinesisClient } from "@aws-sdk/client-kinesis";

/**
 * Put multiple records into a Kinesis stream.
 * @param {{ streamArn: string }} config
 */
export const main = async ({ streamArn }) => {
  const client = new KinesisClient({});
  try {
    await client.send(
      new PutRecordsCommand({
        StreamARN: streamArn,
        Records: [
          {
            Data: new Uint8Array(),
            /**
             * Determines which shard in the stream the data record is assigned to.
             * Partition keys are Unicode strings with a maximum length limit of 256
             * characters for each key. Amazon Kinesis Data Streams uses the
 partition
             * key as input to a hash function that maps the partition key and
             * associated data to a specific shard.
             */
            PartitionKey: "TEST_KEY",
          },
```

```
          {
            Data: new Uint8Array(),
            PartitionKey: "TEST_KEY",
          },
        ],
      }),
    );
  } catch (caught) {
    if (caught instanceof Error) {
      //
    } else {
      throw caught;
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    streamArn: {
      type: "string",
      description: "The ARN of the stream.",
    },
  };

  const { values } = parseArgs({ options });
  main(values);
}
```

- Per i dettagli sull'API, PutRecordsconsulta AWS SDK for JavaScript API Reference.

Esempi serverless

Invocare una funzione Lambda da un trigger Kinesis

L'esempio di codice seguente 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.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. 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

L'esempio di codice seguente mostra come implementare una risposta batch parziale per 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.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. 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 con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Lambda.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Hello Lambda

Il seguente esempio di codice mostra come iniziare a usare Lambda.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Per i dettagli sull'API, ListFunctionsconsulta AWS SDK for JavaScript API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un ruolo IAM e una funzione Lambda, quindi carica il codice del gestore.
- Invocare la funzione con un singolo parametro e ottenere i risultati.
- Aggiorna il codice della funzione e configuralo con una variabile di ambiente.
- Invocare 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, consulta Creare una funzione Lambda con la console.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un ruolo AWS Identity and Access Management (IAM) che conceda a Lambda l'autorizzazione di scrittura nei log.

```
    logger.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);
};
```

Invocare 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,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

Elencare le funzioni per l'account.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Eliminare il ruolo IAM e la funzione Lambda.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
```

```
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Azioni

`CreateFunction`

Il seguente esempio di codice mostra come utilizzare. `CreateFunction`

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const createFunction = async (funcName, roleArn) => {
```

```
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, CreateFunctionconsulta AWS SDK for JavaScript API Reference.

`DeleteFunction`

Il seguente esempio di codice mostra come utilizzare`DeleteFunction`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per i dettagli sull'API, DeleteFunctionconsulta AWS SDK for JavaScript API Reference.

`GetFunction`

Il seguente esempio di codice mostra come utilizzare`GetFunction`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per i dettagli sull'API, GetFunctionconsulta AWS SDK for JavaScript API Reference.

`Invoke`

Il seguente esempio di codice mostra come utilizzare`Invoke`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
```

```
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK for JavaScript .

`ListFunctions`

Il seguente esempio di codice mostra come usare`ListFunctions`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Per i dettagli sull'API, ListFunctionsconsulta AWS SDK for JavaScript API Reference.

`UpdateFunctionCode`

Il seguente esempio di codice mostra come utilizzare`UpdateFunctionCode`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, UpdateFunctionCodeconsulta AWS SDK for JavaScript API Reference.

`UpdateFunctionConfiguration`

Il seguente esempio di codice mostra come utilizzare`UpdateFunctionConfiguration`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
```

```
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

- Per i dettagli sull'API, UpdateFunctionConfigurationconsulta AWS SDK for JavaScript API Reference.

Scenari

Confermare automaticamente gli utenti noti con una funzione Lambda

L'esempio di codice seguente mostra come confermare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PreSignUp`.
- Registra un utente con Amazon Cognito.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e conferma automaticamente gli utenti noti.
- Accedi come nuovo utente, quindi elimina le risorse.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Configura un'esecuzione interattiva dello "Scenario". Gli esempi JavaScript (v3) condividono uno Scenario runner per semplificare esempi complessi. Il codice sorgente completo è attivo. GitHub

```
import { AutoConfirm } from "./scenario-auto-confirm.js";
```

```
/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      UserEmail: "test_email_1@example.com",
    },
    {
      UserName: "test_user_2",
      UserEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      UserEmail: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito
 authentication behavior.",
  });
```

```
}
```

Questo scenario dimostra la conferma automatica di un utente noto. Orchestra i passaggi di esempio.

```
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
} from "./steps-common.js";
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
```

```
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
```

```
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (/** @type {State} */ state) => {
    const [_, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
 "${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (/** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
```

```
    default: (/** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (/** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    if (user) {
      state.errors.push(
        new Error(
          `The user "${state.selectedUser}" already exists in the user pool
 "${state.UserPoolId}".`,
        ),
      );
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
 numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);
```

```
const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (/** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({
        region: state.stackRegion,
        userPoolClientId: state.UserPoolClientId,
        username: state.selectedUser,
        email: state.users.find((u) => u.UserName === state.selectedUser)
          .UserEmail,
        password,
      });

    let [_, err] = await signUp(state.password);

    while (err?.name === "InvalidPasswordException") {
      console.warn("The password you entered was invalid.");
      await createPassword.handle(state);
      [_, err] = await signUp(state.password);
    }

    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
```

```
  async (/** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
    if (logStreamErr) {
      state.errors.push(logStreamErr);
      return;
    }

    console.log(
      `Getting some recent events from log stream "${logStream.logStreamName}"`,
    );
    const [logEvents, logEventsErr] = await getLogEvents({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
      eventCount: 10,
      logStreamName: logStream.logStreamName,
    });
    if (logEventsErr) {
      state.errors.push(logEventsErr);
      return;
    }

    console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
```

```
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });

    if (err?.name === "PasswordResetRequiredException") {
      state.errors.push(new Error("Please reset your password."));
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with: ${state.token.slice(0,
 11)}`,
  { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [_, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
```

```
      state.errors.push(err);
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);

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}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
```

```
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
  );
```

Questi sono passaggi condivisi con altri scenari.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
```

```
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);
```

Un handler per il trigger `PreSignUp` con una funzione Lambda.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "./user-repository";
import { DynamoDBUserRepository } from "./user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user '${event.userName}'`,
```

```
    );

    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }

    const eventEmail = this.getEventUserEmail(event);
    console.log(`Looking up email ${eventEmail}.`);
    const storedUserInfo =
      await this.userRepository.getUserInfoByEmail(eventEmail);

    if (!storedUserInfo) {
      console.log(
        `Email ${eventEmail} not found. Email verification is required.`,
      );
      return event;
    }

    if (storedUserInfo.UserName !== event.userName) {
      console.log(
        `UserEmail ${eventEmail} found, but stored UserName
 '${storedUserInfo.UserName}' does not match supplied UserName '${event.userName}'.
 Verification is required.`,
      );
    } else {
      console.log(
        `UserEmail ${eventEmail} found with matching UserName
 ${storedUserInfo.UserName}. User is confirmed.`,
      );
      event.response.autoConfirmUser = true;
      event.response.autoVerifyEmail = true;
    }
    return event;
  }
}

const createPreSignUpHandler = (): PreSignUpHandler => {
  const tableName = process.env.TABLE_NAME;
  if (!tableName) {
    throw new Error("TABLE_NAME environment variable is not set");
  }

  const userRepository = new DynamoDBUserRepository(tableName);
  return new PreSignUpHandler(userRepository);
```

```
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

Modulo di azioni CloudWatch Logs.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
 unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
```

```
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[] |
 null, unknown]>}
 */
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};
```

Modulo di azioni Amazon Cognito.

```
import {
  AdminGetUserCommand,
```

```
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
```

```
 *   password: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const response = await cognitoClient.send(
      new SignUpCommand({
        ClientId: userPoolClientId,
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 flow.
 * @param {{ region: string, clientId: string, username: string, password: string }}
 config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
```

```
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
```

```
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

Modulo di azioni DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }}
 config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
 null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
```

```
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Creazione di un'applicazione serverless per gestire foto

L'esempio di codice seguente mostra come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per JavaScript (v3)

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (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.

SDK per JavaScript (v3)

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in. GitHub I seguenti estratti mostrano come AWS SDK for JavaScript viene utilizzato all'interno delle funzioni Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });
```

```
  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });
```

```
  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
```

```
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly

- Amazon Textract
- Amazon Translate

Invocazione di una funzione Lambda da un browser

Il seguente esempio di codice mostra come richiamare una AWS Lambda funzione da un browser.

SDK per JavaScript (v3)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente. Questa app utilizza la versione 3. AWS SDK for JavaScript

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

Utilizzo di Gateway API per invocare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da Amazon API Gateway.

SDK per JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando l'API di JavaScript runtime Lambda. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda invocata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

Utilizzo degli eventi pianificati per invocare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

SDK per JavaScript (v3)

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Esempi serverless

Connessione a un database Amazon RDS in una funzione Lambda

L'esempio di codice seguente mostra come implementare una funzione Lambda che si connette a un database RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS 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 database Amazon RDS 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}`)
    };
};
```

Invocare una funzione Lambda da un trigger Kinesis

L'esempio di codice seguente 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.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. 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;
}
```

Invocare una funzione Lambda da un trigger DynamoDB

L'esempio di codice seguente 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.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. 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)}`);
};
```

Invocare una funzione Lambda da un trigger Amazon DocumentDB

L'esempio di codice seguente 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.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. 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));
};
```

Invocare una funzione Lambda da un trigger Amazon MSK

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un cluster Amazon MSK. La funzione recupera il payload MSK e registra il contenuto del record.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Amazon MSK con JavaScript Lambda utilizzando.

```
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)
      })
    }
}
```

Consumo di un evento Amazon MSK con TypeScript Lambda utilizzando.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});
```

```
export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    };
  }
}
```

Invocazione di una funzione Lambda da un trigger Amazon S3

L'esempio di codice seguente 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 evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. 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);
  }
};
```

Invocazione di una funzione Lambda da un trigger Amazon SNS

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dal ricevimento di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento SNS con JavaScript Lambda utilizzando.

```
// 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;
  }
}
```

Consumo di un evento SNS con TypeScript Lambda utilizzando.

```
// 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;
  }
}
```

Invocazione di una funzione Lambda da un trigger Amazon SQS

L'esempio di codice seguente 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.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento SQS con JavaScript Lambda utilizzando.

```
// 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;
  }
}
```

Consumo di un evento SQS con TypeScript Lambda utilizzando.

```
// 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

L'esempio di codice seguente mostra come implementare una risposta batch parziale per 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.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. 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 di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

L'esempio di codice seguente mostra come implementare una risposta batch parziale per 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.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. 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 di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

L'esempio di codice seguente mostra come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. 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 SQS utilizzando Lambda. JavaScript

```
// 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 batch SQS utilizzando Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

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.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un chatbot Amazon Lex

L'esempio di codice seguente mostra come creare un chatbot per coinvolgere i visitatori del sito web.

SDK per JavaScript (v3)

> Mostra come utilizzare l'API Amazon Lex per creare un chatbot all'interno di un'applicazione web per coinvolgere i visitatori del sito web.
>
> Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo Costruire un chatbot Amazon Lex nella guida per gli AWS SDK for JavaScript sviluppatori.
>
> Servizi utilizzati in questo esempio
>
> - Amazon Comprehend
> - Amazon Lex
> - Amazon Translate

Esempi di Amazon Location 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 Location.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Hello Amazon Location

Il seguente esempio di codice mostra come iniziare a utilizzare Amazon Location Service.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  LocationClient,
  ListGeofenceCollectionsCommand,
} from "@aws-sdk/client-location";
```

```
/**
 * Lists geofences from a specified geofence collection asynchronously.
 */
export const main = async () => {
  const region = "eu-west-1";
  const locationClient = new LocationClient({ region: region });
  const listGeofenceCollParams = {
    MaxResults: 100,
  };
  try {
    const command = new ListGeofenceCollectionsCommand(listGeofenceCollParams);
    const response = await locationClient.send(command);
    const geofenceEntries = response.Entries;
    if (geofenceEntries.length === 0) {
      console.log("No Geofences were found in the collection.");
    } else {
      for (const geofenceEntry of geofenceEntries) {
        console.log(`Geofence ID: ${geofenceEntry.CollectionName}`);
      }
    }
  } catch (error) {
    console.error(
      `A validation error occurred while creating geofence: ${error} \n Exiting
 program.`,
    );
    return;
  }
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - ListGeofenceCollections
  - ListGeofences

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Creare una mappa del servizio di posizione Amazon.
- Creare una chiave dell'API del servizio di posizione Amazon.
- Visualizzare l'URL della mappa.
- Creare una collezione di recinzioni geografiche virtuali.
- Memorizzare una geometria di recinzioni geografiche virtuali.
- Creare una risorsa tracciatore.
- Aggiornare la posizione di un dispositivo.
- Recuperare l'aggiornamento più recente della posizione per un dispositivo specificato.
- Creare un calcolatore di route.
- Determinare la distanza tra Seattle e Vancouver.
- Usa il livello superiore di Amazon Location APIs.
- Eliminare gli asset del servizio di posizione Amazon.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
/*
Before running this JavaScript code example, set up your development environment,
 including your credentials.
This demo illustrates how to use the AWS SDK for JavaScript (v3) to work with Amazon
 Location Service.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/getting-
started.html
*/

import {
  Scenario,
  ScenarioAction,
```

```
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

import {
  CreateMapCommand,
  CreateGeofenceCollectionCommand,
  PutGeofenceCommand,
  CreateTrackerCommand,
  BatchUpdateDevicePositionCommand,
  GetDevicePositionCommand,
  CreateRouteCalculatorCommand,
  CalculateRouteCommand,
  LocationClient,
  ConflictException,
  ResourceNotFoundException,
  DeleteGeofenceCollectionCommand,
  DeleteRouteCalculatorCommand,
  DeleteTrackerCommand,
  DeleteMapCommand,
} from "@aws-sdk/client-location";

import {
  GeoPlacesClient,
  ReverseGeocodeCommand,
  SearchNearbyCommand,
  SearchTextCommand,
  GetPlaceCommand,
  ValidationException,
} from "@aws-sdk/client-geo-places";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";

/*The inputs for this example can be edited in the ./input.json.*/
import data from "./inputs.json" with { type: "json" };

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
/* v8 ignore next 3 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
```

```
  verbose: "false",
});

const pressEnterConfirm = new ScenarioInput(
  "confirm",
  "Press Enter to continue",
  {
    type: "confirm",
    verbose: "false",
  },
);

const region = "eu-west-1";

const locationClient = new LocationClient({ region: region });

const greet = new ScenarioOutput(
  "greet",
  "Welcome to the Amazon Location Use demo! \n" +
    "AWS Location Service is a fully managed service offered by Amazon Web Services
 (AWS) that " +
    "provides location-based services for developers. This service simplifies " +
    "the integration of location-based features into applications, making it " +
    "Maps: The service provides access to high-quality maps, satellite imagery, " +
    "and geospatial data from various providers, allowing developers to " +
    "easily embed maps into their applications:\n" +
    "Tracking: The Location Service enables real-time tracking of mobile devices, "
 +
    "assets, or other entities, allowing developers to build applications " +
    "that can monitor the location of people, vehicles, or other objects.\n" +
    "Geocoding: The service provides the ability to convert addresses or " +
    "location names into geographic coordinates (latitude and longitude), " +
    "and vice versa, enabling developers to integrate location-based search " +
    "and routing functionality into their applications. " +
    "Please define values ./inputs.json for each user-defined variable used in this
 app. Otherwise the default is used:\n" +
    "- mapName: The name of the map to be create (default is 'AWSMap').\n" +
    "- keyName: The name of the API key to create (default is ' AWSApiKey')\n" +
    "- collectionName: The name of the geofence collection (default is
 'AWSLocationCollection')\n" +
    "- geoId: The geographic identifier used for the geofence or map (default is
 'geoId')\n" +
    "- trackerName: The name of the tracker (default is 'geoTracker')\n" +
```

```
    "- calculatorName: The name of the route calculator (default is
 'AWSRouteCalc')\n" +
    "- deviceId: The ID of the device (default is 'iPhone-112356')",

  { header: true },
);
const displayCreateAMap = new ScenarioOutput(
  "displayCreateAMap",
  "1. Create a map\n" +
    "An AWS Location map can enhance the user experience of your " +
    " application by providing accurate and personalized location-based " +
    " features. For example, you could use the geocoding capabilities to " +
    " allow users to search for and locate businesses, landmarks, or " +
    " other points of interest within a specific region.",
);

const sdkCreateAMap = new ScenarioAction(
  "sdkCreateAMap",
  async (/** @type {State} */ state) => {
    const createMapParams = {
      MapName: `${data.inputs.mapName}`,
      Configuration: { style: "VectorEsriNavigation" },
    };
    try {
      const command = new CreateMapCommand(createMapParams);
      const response = await locationClient.send(command);
      state.MapName = response.MapName;
      console.log("Map created. Map ARN is: ", state.MapName);
    } catch (error) {
      console.error("Error creating map: ", error);
      throw error;
    }
  },
);

const displayMapUrl = new ScenarioOutput(
  "displayMapUrl",
  "2. Display Map URL\n" +
    "When you embed a map in a web app or website, the API key is " +
    "included in the map tile URL to authenticate requests. You can " +
    "restrict API keys to specific AWS Location operations (e.g., only " +
    "maps, not geocoding). API keys can expire, ensuring temporary " +
    "access control.\n" +
    "In order to get the MAP URL you need to create and get the API Key value. " +
```

```
    "You can create and get the key value using the AWS Management Console under " +
    "Location Services. These operations cannot be completed using the " +
    "AWS SDK. For more information about getting the key value, see " +
    "the AWS Location Documentation.",
);

const sdkDisplayMapUrl = new ScenarioAction(
  "sdkDisplayMapUrl",
  async (/** @type {State} */ state) => {
    const mapURL = `https://maps.geo.aws.amazon.com/maps/v0/maps/${state.MapName}/
tiles/{z}/{x}/{y}?key=API_KEY_VALUE`;
    state.mapURL = mapURL;
    console.log(
      `Replace \'API_KEY_VALUE\' in the following URL with the value for the API key
 you create and get from the AWS Management Console under Location Services. This is
 then the Map URL you can embed this URL in your Web app:\n
${state.mapURL}`,
    );
  },
);
const displayCreateGeoFenceColl = new ScenarioOutput(
  "displayCreateGeoFenceColl",
  "3. Create a geofence collection, which manages and stores geofences.",
);

const sdkCreateGeoFenceColl = new ScenarioAction(
  "sdkCreateGeoFenceColl",
  async (/** @type {State} */ state) => {
    // Creates a new geofence collection.
    const geoFenceCollParams = {
      CollectionName: `${data.inputs.collectionName}`,
    };
    try {
      const command = new CreateGeofenceCollectionCommand(geoFenceCollParams);
      const response = await locationClient.send(command);
      state.CollectionName = response.CollectionName;
      console.log(
        `The geofence collection was successfully created: ${state.CollectionName}`,
      );
    } catch (caught) {
      if (caught instanceof ConflictException) {
        console.error(
          `An unexpected error occurred while creating the geofence collection:
 ${caught.message} \n Exiting program.`,
```

```
        );
        return;
      }
    }
  },
);
const displayStoreGeometry = new ScenarioOutput(
  "displayStoreGeometry",
  "4. Store a geofence geometry in a given geofence collection. " +
    "An AWS Location geofence is a virtual boundary that defines a geographic area "
 +
    "on a map. It is a useful feature for tracking the location of " +
    "assets or monitoring the movement of objects within a specific region. " +
    "To define a geofence, you need to specify the coordinates of a " +
    "polygon that represents the area of interest. The polygon must be " +
    "defined in a counter-clockwise direction, meaning that the points of " +
    "the polygon must be listed in a counter-clockwise order. " +
    "This is a requirement for the AWS Location service to correctly " +
    "interpret the geofence and ensure that the location data is " +
    "accurately processed within the defined area.",
);

const sdkStoreGeometry = new ScenarioAction(
  "sdkStoreGeometry",
  async (/** @type {State} */ state) => {
    const geoFenceGeoParams = {
      CollectionName: `${data.inputs.collectionName}`,
      GeofenceId: `${data.inputs.geoId}`,
      Geometry: {
        Polygon: [
          [
            [-122.3381, 47.6101],
            [-122.3281, 47.6101],
            [-122.3281, 47.6201],
            [-122.3381, 47.6201],
            [-122.3381, 47.6101],
          ],
        ],
      },
    };
    try {
      const command = new PutGeofenceCommand(geoFenceGeoParams);
      const response = await locationClient.send(command);
      state.GeoFencId = response.GeofenceId;
```

```
      console.log("GeoFence created. GeoFence ID is: ", state.GeoFencId);
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
          `A validation error occurred while creating geofence: ${caught.message} \n
 Exiting program.`,
        );
        return;
      }
    }
  },
);
const displayCreateTracker = new ScenarioOutput(
  "displayCreateTracker",
  "5. Create a tracker resource which lets you retrieve current and historical
 location of devices.",
);

const sdkCreateTracker = new ScenarioAction(
  "sdkCreateTracker",
  async (/** @type {State} */ state) => {
    //Creates a new tracker resource in your AWS account, which you can use to track
 the location of devices.
    const createTrackerParams = {
      TrackerName: `${data.inputs.trackerName}`,
      Description: "Created using the JavaScript V3 SDK",
      PositionFiltering: "TimeBased",
    };
    try {
      const command = new CreateTrackerCommand(createTrackerParams);
      const response = await locationClient.send(command);
      state.trackerName = response.TrackerName;
      console.log("Tracker created. Tracker name is : ", state.trackerName);
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `A validation error occurred while creating geofence: ${caught.message} \n
 Exiting program.`,
        );
      } else {
        `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
      }
      return;
    }
```

```
  },
);
const displayUpdatePosition = new ScenarioOutput(
  "displayUpdatePosition",
  "6. Update the position of a device in the location tracking system." +
    "The AWS Location Service does not enforce a strict format for deviceId, but it
 must:\n " +
    "- Be a string (case-sensitive).\n" +
    "- Be 1-100 characters long.\n" +
    "- Contain only: Alphanumeric characters (A-Z, a-z, 0-9); Underscores (_);
 Hyphens (-); and be the same ID used when sending and retrieving positions.",
);

const sdkUpdatePosition = new ScenarioAction(
  "sdkUpdatePosition",
  async (/** @type {State} */ state) => {
    // Updates the position of a device in the location tracking system.

    const updateDevicePosParams = {
      TrackerName: `${data.inputs.trackerName}`,
      Updates: [
        {
          DeviceId: `${data.inputs.deviceId}`,
          SampleTime: new Date(),
          Position: [-122.4194, 37.7749],
        },
      ],
    };
    try {
      const command = new BatchUpdateDevicePositionCommand(
        updateDevicePosParams,
      );
      const response = await locationClient.send(command);
      console.log(
        `Device with id ${data.inputs.deviceId} was successfully updated in the
 location tracking system. `,
      );
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `A validation error occurred while updating the device: ${caught.message}
 \n Exiting program.`,
        );
      }
```

```
    }
  },
);
const displayRetrievePosition = new ScenarioOutput(
  "displayRetrievePosition",
  "7. Retrieve the most recent position update for a specified device.",
);

const sdkRetrievePosition = new ScenarioAction(
  "sdkRetrievePosition",
  async (/** @type {State} */ state) => {
    const devicePositionParams = {
      TrackerName: `${data.inputs.trackerName}`,
      DeviceId: `${data.inputs.deviceId}`,
    };
    try {
      const command = new GetDevicePositionCommand(devicePositionParams);
      const response = await locationClient.send(command);
      state.position = response.Position;
      console.log("Successfully fetched device position: : ", state.position);
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `"The resource was not found: ${caught.message} \n Exiting program.`,
        );
      } else {
        `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
      }
      return;
    }
  },
);
const displayCreateRouteCalc = new ScenarioOutput(
  "displayCreateRouteCalc",
  "8. Create a route calculator.",
);

const sdkCreateRouteCalc = new ScenarioAction(
  "sdkCreateRouteCalc",
  async (/** @type {State} */ state) => {
    const routeCalcParams = {
      CalculatorName: `${data.inputs.calculatorName}`,
      DataSource: "Esri",
    };
```

```
    try {
      // Creates a new route calculator with the specified name and data source.
      const command = new CreateRouteCalculatorCommand(routeCalcParams);
      const response = await locationClient.send(command);
      state.CalculatorName = response.CalculatorName;
      console.log(
        "Route calculator created successfully. Calculator name is: ",
        state.CalculatorName,
      );
    } catch (caught) {
      if (caught instanceof ConflictException) {
        console.error(
          `An conflict occurred: ${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
  },
);
const displayDetermineDist = new ScenarioOutput(
  "displayDetermineDist",
  "9. Determine the distance between Seattle and Vancouver using the route
 calculator.",
);

const sdkDetermineDist = new ScenarioAction(
  "sdkDetermineDist",
  async (/** @type {State} */ state) => {
    // Calculates the distance between two locations asynchronously.
    const determineDist = {
      CalculatorName: `${data.inputs.calculatorName}`,
      DeparturePosition: [-122.3321, 47.6062],
      DestinationPosition: [-123.1216, 49.2827],
      TravelMode: "Car",
      DistanceUnit: "Kilometers",
    };
    try {
      const command = new CalculateRouteCommand(determineDist);
      const response = await locationClient.send(command);

      console.log(
        "Successfully calculated route. The distance in kilometers is : ",
        response.Summary.Distance,
      );
```

```
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `Failed to calculate route: ${caught.message} \n Exiting program.`,
        );
      }
      return;
    }
  },
);
const displayUseGeoPlacesClient = new ScenarioOutput(
  "displayUseGeoPlacesClient",
  "10. Use the GeoPlacesAsyncClient to perform additional operations. " +
    "This scenario will show use of the GeoPlacesClient that enables" +
    "location search and geocoding capabilities for your applications. " +
    "We are going to use this client to perform these AWS Location tasks: \n" +
    " - Reverse Geocoding (reverseGeocode): Converts geographic coordinates into
 addresses.\n " +
    " - Place Search (searchText): Finds places based on search queries.\n " +
    " - Nearby Search (searchNearby): Finds places near a specific location.\n " +
    "First we will perform a Reverse Geocoding operation",
);

const sdkUseGeoPlacesClient = new ScenarioAction(
  "sdkUseGeoPlacesClient",
  async (/** @type {State} */ state) => {
    const geoPlacesClient = new GeoPlacesClient({ region: region });

    const reverseGeoCodeParams = {
      QueryPosition: [-122.4194, 37.7749],
    };
    const searchTextParams = {
      QueryText: "coffee shop",
      BiasPosition: [-122.4194, 37.7749], //San Fransisco
    };
    const searchNearbyParams = {
      QueryPosition: [-122.4194, 37.7749],
      QueryRadius: Number("1000"),
    };
    try {
      /*   Performs reverse geocoding using the AWS Geo Places API.
     Reverse geocoding is the process of converting geographic coordinates (latitude
 and longitude) to a human-readable address.
```

```
    This method uses the latitude and longitude of San Francisco as the input, and
prints the resulting address.*/

     console.log("Use latitude 37.7749 and longitude -122.4194.");
     const command = new ReverseGeocodeCommand(reverseGeoCodeParams);
     const response = await geoPlacesClient.send(command);
     console.log(
       "Successfully calculated route. The distance in kilometers is : ",
       response.ResultItems[0].Distance,
     );
   } catch (caught) {
     if (caught instanceof ValidationException) {
       console.error(
         `An conflict occurred: ${caught.message} \n Exiting program.`,
       );
       return;
     }
   }
   try {
     console.log(
       "Now we are going to perform a text search using coffee shop",
     );

     /*Searches for a place using the provided search query and prints the detailed
information of the first result.
 @param searchTextParams the search query to be used for the place search (ex,
coffee shop)*/

     const command = new SearchTextCommand(searchTextParams);
     const response = await geoPlacesClient.send(command);
     const placeId = response.ResultItems[0].PlaceId.toString();
     const getPlaceCommand = new GetPlaceCommand({
       PlaceId: placeId,
     });
     const getPlaceResponse = await geoPlacesClient.send(getPlaceCommand);
     console.log(
       `Detailed Place Information: \n Name and address:
${getPlaceResponse.Address.Label}`,
     );

     const foodTypes = getPlaceResponse.FoodTypes;
     if (foodTypes.length) {
       console.log("Food Types: ");
       for (const foodType of foodTypes) {
```

```
          console.log("- ", foodType.LocalizedName);
        }
      } else {
        console.log("No food types available.");
      }
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
          `An conflict occurred: ${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
    try {
      console.log("\nNow we are going to perform a nearby search.");
      const command = new SearchNearbyCommand(searchNearbyParams);
      const response = await geoPlacesClient.send(command);
      const resultItems = response.ResultItems;
      console.log("\nSuccessfully performed nearby search.");
      for (const resultItem of resultItems) {
        console.log("Name and address: ", resultItem.Address.Label);
        console.log("Distance: ", resultItem.Distance);
      }
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
          `An conflict occurred: ${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
  },
);

const displayDeleteResources = new ScenarioOutput(
  "displayDeleteResources",
  "11. Delete the AWS Location Services resources. " +
    "Would you like to delete the AWS Location Services resources? (y/n)",
);

const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    const deleteGeofenceCollParams = {
```

```
      CollectionName: `${state.CollectionName}`,
    };
    const deleteRouteCalculatorParams = {
      CalculatorName: `${state.CalculatorName}`,
    };
    const deleteTrackerParams = { TrackerName: `${state.trackerName}` };
    const deleteMapParams = { MapName: `${state.MapName}` };
    try {
      const command = new DeleteMapCommand(deleteMapParams);
      const response = await locationClient.send(command);
      console.log("Map deleted.");
    } catch (error) {
      console.log("Error deleting map: ", error);
    }
    try {
      const command = new DeleteGeofenceCollectionCommand(
        deleteGeofenceCollParams,
      );
      const response = await locationClient.send(command);
      console.log("Geofence collection deleted.");
    } catch (error) {
      console.log("Error deleting geofence collection: ", error);
    }
    try {
      const command = new DeleteRouteCalculatorCommand(
        deleteRouteCalculatorParams,
      );
      const response = await locationClient.send(command);
      console.log("Route calculator deleted.");
    } catch (error) {
      console.log("Error deleting route calculator: ", error);
    }
    try {
      const command = new DeleteTrackerCommand(deleteTrackerParams);
      const response = await locationClient.send(command);
      console.log("Tracker deleted.");
    } catch (error) {
      console.log("Error deleting tracker: ", error);
    }
  },
);

const goodbye = new ScenarioOutput(
  "goodbye",
```

```
  "Thank you for checking out the Amazon Location Service Use demo. We hope you " +
    "learned something new, or got some inspiration for your own apps today!" +
    " For more Amazon Location Services examples in different programming languages,
 have a look at: " +
    "https://docs.aws.amazon.com/code-library/latest/ug/
location_code_examples.html",
);

const myScenario = new Scenario("Location Services Scenario", [
  greet,
  pressEnter,
  displayCreateAMap,
  sdkCreateAMap,
  pressEnter,
  displayMapUrl,
  sdkDisplayMapUrl,
  pressEnter,
  displayCreateGeoFenceColl,
  sdkCreateGeoFenceColl,
  pressEnter,
  displayStoreGeometry,
  sdkStoreGeometry,
  pressEnter,
  displayCreateTracker,
  sdkCreateTracker,
  pressEnter,
  displayUpdatePosition,
  sdkUpdatePosition,
  pressEnter,
  displayRetrievePosition,
  sdkRetrievePosition,
  pressEnter,
  displayCreateRouteCalc,
  sdkCreateRouteCalc,
  pressEnter,
  displayDetermineDist,
  sdkDetermineDist,
  pressEnter,
  displayUseGeoPlacesClient,
  sdkUseGeoPlacesClient,
  pressEnter,
  displayDeleteResources,
  pressEnterConfirm,
  sdkDeleteResources,
```

```
  goodbye,
]);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

Azioni

`BatchUpdateDevicePosition`

Il seguente esempio di codice mostra come usare`BatchUpdateDevicePosition`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  BatchUpdateDevicePositionCommand,
  LocationClient,
  ResourceNotFoundException,
```

```
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };
const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });
const updateDevicePosParams = {
  TrackerName: `${data.inputs.trackerName}`,
  Updates: [
    {
      DeviceId: `${data.inputs.deviceId}`,
      SampleTime: new Date(),
      Position: [-122.4194, 37.7749],
    },
  ],
};
export const main = async () => {
  try {
    const command = new BatchUpdateDevicePositionCommand(updateDevicePosParams);
    const response = await locationClient.send(command);
    //console.log("response ", response.Errors[0].Error);

    console.log(
      `Device with id ${data.inputs.deviceId} was successfully updated in the
 location tracking system. `,
      response,
    );
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Per i dettagli sull'API, BatchUpdateDevicePositionconsulta AWS SDK for JavaScript API Reference.

`CalculateRoute`

Il seguente esempio di codice mostra come utilizzare`CalculateRoute`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  CalculateRouteCommand,
  ResourceNotFoundException,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });

export const main = async () => {
  const routeCalcParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
    DeparturePosition: [-122.3321, 47.6062],
    DestinationPosition: [-123.1216, 49.2827],
    TravelMode: "Car",
    DistanceUnit: "Kilometers",
  };
  try {
    const command = new CalculateRouteCommand(routeCalcParams);
    const response = await locationClient.send(command);

    console.log(
      "Successfully calculated route. The distance in kilometers is : ",
      response.Summary.Distance,
    );
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `An conflict occurred: ${caught.message} \n Exiting program.`,
      );
      return;
    }
```

```
  }
};
```

- Per i dettagli sull'API, CalculateRouteconsulta AWS SDK for JavaScript API Reference.

`CreateGeofenceCollection`

Il seguente esempio di codice mostra come utilizzare`CreateGeofenceCollection`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  ConflictException,
  CreateGeofenceCollectionCommand,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const geoFenceCollParams = {
    CollectionName: `${data.inputs.collectionName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateGeofenceCollectionCommand(geoFenceCollParams);
    const response = await locationClient.send(command);
    console.log(
      "Collection created. Collection name is: ",
      response.CollectionName,
    );
```

```
  } catch (caught) {
    if (caught instanceof ConflictException) {
      console.error("A conflict occurred. Exiting program.");
      return;
    }
  }
};
```

- Per i dettagli sull'API, CreateGeofenceCollectionconsulta AWS SDK for JavaScript API Reference.

`CreateMap`

Il seguente esempio di codice mostra come utilizzare`CreateMap`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import { CreateMapCommand, LocationClient } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const CreateMapCommandInput = {
    MapName: `${data.inputs.mapName}`,
    Configuration: { style: "VectorEsriNavigation" },
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateMapCommand(CreateMapCommandInput);
    const response = await locationClient.send(command);
    console.log("Map created. Map ARN is : ", response.MapArn);
```

```
  } catch (error) {
    console.error("Error creating map: ", error);
    throw error;
  }
};
```

- Per i dettagli sull'API, CreateMapconsulta AWS SDK for JavaScript API Reference.

`CreateRouteCalculator`

Il seguente esempio di codice mostra come utilizzare`CreateRouteCalculator`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  ConflictException,
  CreateRouteCalculatorCommand,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });

export const main = async () => {
  const routeCalcParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
    DataSource: "Esri",
  };
  try {
    const command = new CreateRouteCalculatorCommand(routeCalcParams);
    const response = await locationClient.send(command);
```

```
    console.log(
      "Route calculator created successfully. Calculator name is ",
      response.CalculatorName,
    );
  } catch (caught) {
    if (caught instanceof ConflictException) {
      console.error(
        `An conflict occurred: ${caught.message} \n Exiting program.`,
      );
      return;
    }
  }
};
```

- Per i dettagli sull'API, CreateRouteCalculatorconsulta AWS SDK for JavaScript API Reference.

`CreateTracker`

Il seguente esempio di codice mostra come utilizzare`CreateTracker`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import { CreateTrackerCommand, LocationClient } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const createTrackerParams = {
    TrackerName: `${data.inputs.trackerName}`,
  };
  const locationClient = new LocationClient({ region: region });
```

```
  try {
    const command = new CreateTrackerCommand(createTrackerParams);
    const response = await locationClient.send(command);
    //state.trackerName - response.TrackerName;
    console.log("Tracker created. Tracker name is : ", response.TrackerName);
  } catch (error) {
    console.error("Error creating map: ", error);
    throw error;
  }
};
```

- Per i dettagli sull'API, CreateTrackerconsulta AWS SDK for JavaScript API Reference.

`DeleteGeofenceCollection`

Il seguente esempio di codice mostra come utilizzare`DeleteGeofenceCollection`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  DeleteGeofenceCollectionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteGeofenceCollParams = {
    CollectionName: `${data.inputs.collectionName}`,
  };
  const locationClient = new LocationClient({ region: region });
```

```
  try {
    const command = new DeleteGeofenceCollectionCommand(
      deleteGeofenceCollParams,
    );
    const response = await locationClient.send(command);
    console.log("Collection deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `${data.inputs.collectionName} Geofence collection not found.`,
      );
      return;
    }
  }
};
```

- Per i dettagli sull'API, DeleteGeofenceCollectionconsulta AWS SDK for JavaScript API Reference.

`DeleteMap`

Il seguente esempio di codice mostra come utilizzare`DeleteMap`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  DeleteMapCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };
```

```
const region = "eu-west-1";

export const main = async () => {
  const deleteMapParams = {
    MapName: `${data.inputs.mapName}`,
  };
  try {
    const locationClient = new LocationClient({ region: region });
    const command = new DeleteMapCommand(deleteMapParams);
    const response = await locationClient.send(command);
    console.log("Map deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(`${data.inputs.mapName} map not found.`);
      return;
    }
  }
};
```

- Per i dettagli sull'API, DeleteMapconsulta AWS SDK for JavaScript API Reference.

`DeleteRouteCalculator`

Il seguente esempio di codice mostra come utilizzare`DeleteRouteCalculator`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  DeleteRouteCalculatorCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };
```

```
const region = "eu-west-1";

export const main = async () => {
  const deleteRouteCalculatorParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
  };
  try {
    const locationClient = new LocationClient({ region: region });
    const command = new DeleteRouteCalculatorCommand(
      deleteRouteCalculatorParams,
    );
    const response = await locationClient.send(command);
    console.log("Route calculator deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `${data.inputs.calculatorName} route calculator not found.`,
      );
      return;
    }
  }
};
```

- Per i dettagli sull'API, DeleteRouteCalculatorconsulta AWS SDK for JavaScript API Reference.

`DeleteTracker`

Il seguente esempio di codice mostra come utilizzare`DeleteTracker`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
```

```
  DeleteTrackerCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteTrackerParams = {
    TrackerName: `${data.inputs.trackerName}`,
  };
  try {
    const locationClient = new LocationClient({ region: region });
    const command = new DeleteTrackerCommand(deleteTrackerParams);
    const response = await locationClient.send(command);
    console.log("Tracker deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(`${data.inputs.trackerName} tracker not found.`);
      return;
    }
  }
};
```

- Per i dettagli sull'API, DeleteTrackerconsulta AWS SDK for JavaScript API Reference.

`GetDevicePosition`

Il seguente esempio di codice mostra come utilizzare`GetDevicePosition`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
```

```
import {
  GetDevicePositionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const locationClient = new LocationClient({ region: region });
  const deviceId = `${data.inputs.deviceId}`;
  const trackerName = `${data.inputs.trackerName}`;

  const devicePositionParams = {
    DeviceId: deviceId,
    TrackerName: trackerName,
  };
  try {
    const command = new GetDevicePositionCommand(devicePositionParams);
    const response = await locationClient.send(command);
    //state.position = response.position;
    console.log("Successfully fetched device position: ", response);
  } catch (error) {
    console.log("Error ", error);
    /*  if (caught instanceof ResourceNotFoundException) {
      console.error(
        `"The resource was not found: ${caught.message} \n Exiting program.`,
      );
    } else {
      `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
    }
    return;*/
  }
};
```

- Per i dettagli sull'API, GetDevicePositionconsulta AWS SDK for JavaScript API Reference.

`PutGeofence`

Il seguente esempio di codice mostra come utilizzare`PutGeofence`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { fileURLToPath } from "node:url";
import {
  PutGeofenceCommand,
  LocationClient,
  ValidationException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });
export const main = async () => {
  const geoFenceGeoParams = {
    CollectionName: `${data.inputs.collectionName}`,
    GeofenceId: `${data.inputs.geoId}`,
    Geometry: {
      Polygon: [
        [
          [-122.3381, 47.6101],
          [-122.3281, 47.6101],
          [-122.3281, 47.6201],
          [-122.3381, 47.6201],
          [-122.3381, 47.6101],
        ],
      ],
    },
  };
  try {
    const command = new PutGeofenceCommand(geoFenceGeoParams);
    const response = await locationClient.send(command);
    console.log("GeoFence created. GeoFence ID is: ", response.GeofenceId);
  } catch (error) {
    console.error(
      `A validation error occurred while creating geofence: ${error} \n Exiting
 program.`,
```

```
    );
    return;
  }
};
```

- Per i dettagli sull'API, PutGeofenceconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon MSK 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 MSK.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Esempi serverless

Esempi serverless

Invocare una funzione Lambda da un trigger Amazon MSK

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un cluster Amazon MSK. La funzione recupera il payload MSK e registra il contenuto del record.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Amazon MSK con JavaScript Lambda utilizzando.

```
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)
      })
    }
}
```

Consumo di un evento Amazon MSK con TypeScript Lambda utilizzando.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
```

```
        logger.error('Error processing event', { error });
        throw error;
      }
    };
  }
}
```

Esempi di Amazon Personalize utilizzando SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Personalize.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`CreateBatchInferenceJob`

Il seguente esempio di codice mostra come usare. `CreateBatchInferenceJob`

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
```

```
import { CreateBatchInferenceJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch inference job's parameters.

export const createBatchInferenceJobParam = {
  jobName: "JOB_NAME",
  jobInput: {
    s3DataSource: {
      path: "INPUT_PATH",
    },
  },
  jobOutput: {
    s3DataDestination: {
      path: "OUTPUT_PATH",
    },
  },
  roleArn: "ROLE_ARN",
  solutionVersionArn: "SOLUTION_VERSION_ARN",
  numResults: 20,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateBatchInferenceJobCommand(createBatchInferenceJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateBatchInferenceJobconsulta AWS SDK for JavaScript API Reference.

`CreateBatchSegmentJob`

Il seguente esempio di codice mostra come utilizzare`CreateBatchSegmentJob`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: "NAME",
  jobInput: {
    s3DataSource: {
      path: "INPUT_PATH",
    },
  },
  jobOutput: {
    s3DataDestination: {
      path: "OUTPUT_PATH",
    },
  },
  roleArn: "ROLE_ARN",
  solutionVersionArn: "SOLUTION_VERSION_ARN",
  numResults: 20,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateBatchSegmentJobCommand(createBatchSegmentJobParam),
    );
```

```
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateBatchSegmentJobconsulta AWS SDK for JavaScript API Reference.

`CreateCampaign`

Il seguente esempio di codice mostra come utilizzare`CreateCampaign`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.

import { CreateCampaignCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the campaign's parameters.
export const createCampaignParam = {
  solutionVersionArn: "SOLUTION_VERSION_ARN" /* required */,
  name: "NAME" /* required */,
  minProvisionedTPS: 1 /* optional integer */,
};

export const run = async () => {
  try {
```

```
    const response = await personalizeClient.send(
      new CreateCampaignCommand(createCampaignParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateCampaignconsulta AWS SDK for JavaScript API Reference.

`CreateDataset`

Il seguente esempio di codice mostra come utilizzare`CreateDataset`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset's parameters.
export const createDatasetParam = {
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  datasetType: "DATASET_TYPE" /* required */,
  name: "NAME" /* required */,
  schemaArn: "SCHEMA_ARN" /* required */,
};

export const run = async () => {
```

```
  try {
    const response = await personalizeClient.send(
      new CreateDatasetCommand(createDatasetParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateDatasetconsulta AWS SDK for JavaScript API Reference.

`CreateDatasetExportJob`

Il seguente esempio di codice mostra come utilizzare`CreateDatasetExportJob`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetExportJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the export job parameters.
export const datasetExportJobParam = {
  datasetArn: "DATASET_ARN" /* required */,
  jobOutput: {
    s3DataDestination: {
      path: "S3_DESTINATION_PATH" /* required */,
      //kmsKeyArn: 'ARN'  /* include if your bucket uses AWS KMS for encryption
    },
```

```
  },
  jobName: "NAME" /* required */,
  roleArn: "ROLE_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetExportJobCommand(datasetExportJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateDatasetExportJobconsulta AWS SDK for JavaScript API Reference.

`CreateDatasetGroup`

Il seguente esempio di codice mostra come utilizzare`CreateDatasetGroup`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.

import { 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 del dominio.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetGroupCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the domain dataset group parameters.
export const domainDatasetGroupParams = {
  name: "NAME" /* required */,
  domain:
    "DOMAIN" /* required for a domain dsg, specify ECOMMERCE or VIDEO_ON_DEMAND */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetGroupCommand(domainDatasetGroupParams),
    );
    console.log("Success", response);
    return response; // For unit tests.
```

```
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateDatasetGroupconsulta AWS SDK for JavaScript API Reference.

`CreateDatasetImportJob`

Il seguente esempio di codice mostra come utilizzare`CreateDatasetImportJob`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetImportJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset import job parameters.
export const datasetImportJobParam = {
  datasetArn: "DATASET_ARN" /* required */,
  dataSource: {
    /* required */
    dataLocation: "S3_PATH",
  },
  jobName: "NAME" /* required */,
  roleArn: "ROLE_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
```

```
      new CreateDatasetImportJobCommand(datasetImportJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateDatasetImportJobconsulta AWS SDK for JavaScript API Reference.

`CreateEventTracker`

Il seguente esempio di codice mostra come utilizzare`CreateEventTracker`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateEventTrackerCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the event tracker's parameters.
export const createEventTrackerParam = {
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  name: "NAME" /* required */,
};

export const run = async () => {
  try {
```

```
    const response = await personalizeClient.send(
      new CreateEventTrackerCommand(createEventTrackerParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateEventTrackerconsulta AWS SDK for JavaScript API Reference.

`CreateFilter`

Il seguente esempio di codice mostra come utilizzare`CreateFilter`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateFilterCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the filter's parameters.
export const createFilterParam = {
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  name: "NAME" /* required */,
  filterExpression: "FILTER_EXPRESSION" /*required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
```

```
      new CreateFilterCommand(createFilterParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateFilterconsulta AWS SDK for JavaScript API Reference.

`CreateRecommender`

Il seguente esempio di codice mostra come utilizzare`CreateRecommender`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateRecommenderCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the recommender's parameters.
export const createRecommenderParam = {
  name: "NAME" /* required */,
  recipeArn: "RECIPE_ARN" /* required */,
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
```

```
      new CreateRecommenderCommand(createRecommenderParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateRecommenderconsulta AWS SDK for JavaScript API Reference.

`CreateSchema`

Il seguente esempio di codice mostra come utilizzare`CreateSchema`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from "node:fs";

const 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 "node:fs";

const schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = "TEST"; // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: "NAME" /* required */,
```

```
  schema: mySchema /* required */,
  domain:
    "DOMAIN" /* required for a domain dataset group, specify ECOMMERCE or
 VIDEO_ON_DEMAND */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSchemaCommand(createDomainSchemaParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateSchemaconsulta AWS SDK for JavaScript API Reference.

`CreateSolution`

Il seguente esempio di codice mostra come utilizzare`CreateSolution`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution parameters.
export const createSolutionParam = {
```

```
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  recipeArn: "RECIPE_ARN" /* required */,
  name: "NAME" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSolutionCommand(createSolutionParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateSolutionconsulta AWS SDK for JavaScript API Reference.

`CreateSolutionVersion`

Il seguente esempio di codice mostra come utilizzare`CreateSolutionVersion`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionVersionCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution version parameters.
export const solutionVersionParam = {
  solutionArn: "SOLUTION_ARN" /* required */,
```

```
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSolutionVersionCommand(solutionVersionParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, CreateSolutionVersionconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon Personalize Events utilizzando SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Personalize Events.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`PutEvents`

Il seguente esempio di codice mostra come usare. `PutEvents`

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
 Replace it with your sentAt timestamp in UNIX format.

// Set put events parameters.
const putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
```

```
  }
};
run();
```

- Per i dettagli sull'API, PutEventsconsulta AWS SDK for JavaScript API Reference.

`PutItems`

Il seguente esempio di codice mostra come utilizzare`PutItems`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { 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.
const putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /*  required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
 "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
    },
  ],
};
export const run = async () => {
  try {
```

```
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, PutItemsconsulta AWS SDK for JavaScript API Reference.

`PutUsers`

Il seguente esempio di codice mostra come utilizzare`PutUsers`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { 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.
const putUsersParam = {
  datasetArn: "DATASET_ARN",
  users: [
    {
      userId: "USER_ID",
      properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
    },
```

```
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, PutUsersconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon Personalize Runtime utilizzando SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Personalize Runtime.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`GetPersonalizedRanking`

Il seguente esempio di codice mostra come usare. `GetPersonalizedRanking`

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { GetPersonalizedRankingCommand } from "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the ranking request parameters.
export const getPersonalizedRankingParam = {
  campaignArn: "CAMPAIGN_ARN" /* required */,
  userId: "USER_ID" /* required */,
  inputList: ["ITEM_ID_1", "ITEM_ID_2", "ITEM_ID_3", "ITEM_ID_4"],
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetPersonalizedRankingCommand(getPersonalizedRankingParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, GetPersonalizedRankingconsulta AWS SDK for JavaScript API Reference.

`GetRecommendations`

Il seguente esempio di codice mostra come utilizzare`GetRecommendations`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from "@aws-sdk/client-personalize-runtime";

import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: "CAMPAIGN_ARN" /* required */,
  userId: "USER_ID" /* required */,
  numResults: 15 /* optional */,
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetRecommendationsCommand(getRecommendationsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Ottiene una raccomandazione 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();
```

Ottiene un elenco di raccomandazioni filtrate da un programma di raccomandazione creato in un gruppo di set di dati di un dominio.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here:
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: "CAMPAIGN_ARN" /* required */,
```

```
  userId: "USER_ID" /* required */,
  numResults: 15 /* optional */,
  filterArn: "FILTER_ARN" /* required to filter recommendations */,
  filterValues: {
    PROPERTY:
      '"VALUE"' /* Only required if your filter has a placeholder parameter */,
  },
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetRecommendationsCommand(getRecommendationsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, GetRecommendationsconsulta AWS SDK for JavaScript API Reference.

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 azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`SendMessages`

Il seguente esempio di codice mostra come usare. `SendMessages`

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
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.
const subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
const 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.
const 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.
const 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);
    }
    console.log(recipientResult.MessageId);
  } catch (err) {
    console.log(err.message);
  }
};

run();
```

Invia un messaggio SMS.

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "./libs/pinClient.js";

/* 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.*/
const messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
const 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.*/

const senderId = "MySenderID";

// Specify the parameters to pass to the API.
const 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 i dettagli sull'API, SendMessagesconsulta AWS SDK for JavaScript API Reference.

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.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Scenari

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.

SDK per (v3 JavaScript)

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 RDS 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.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

L'esempio di codice seguente mostra come creare un'applicazione web che traccia gli elementi di lavoro in database Amazon Aurora serverless e utilizza Amazon Simple Email Service (Amazon SES) per inviare report.

SDK per (v3 JavaScript)

> Mostra come utilizzare AWS SDK for JavaScript (v3) per creare un'applicazione Web che tenga traccia degli elementi di lavoro in un database Amazon Aurora e invii report tramite e-mail utilizzando Amazon Simple Email Service (Amazon SES). Questo esempio utilizza un front-end creato con React.js per interagire con un backend Express Node.js.
>
> - Integra un'applicazione web React.js con. Servizi AWS
> - Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
> - Invia un report per e-mail degli elementi di lavoro filtrati tramite Amazon SES.
> - Distribuisci e gestisci risorse di esempio con lo AWS CloudFormation script incluso.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Aurora
- Amazon RDS
- Servizi di dati di Amazon RDS
- Amazon SES

Esempi serverless

Connessione a un database Amazon RDS in una funzione Lambda

L'esempio di codice seguente mostra come implementare una funzione Lambda che si connette a un database RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS 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 database Amazon RDS 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.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Scenari

Scenari

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

L'esempio di codice seguente mostra come creare un'applicazione web che traccia gli elementi di lavoro in database Amazon Aurora serverless e utilizza Amazon Simple Email Service (Amazon SES) per inviare report.

SDK per (v3 JavaScript)

Mostra come utilizzare AWS SDK for JavaScript (v3) per creare un'applicazione Web che tenga traccia degli elementi di lavoro in un database Amazon Aurora e invii report tramite e-mail utilizzando Amazon Simple Email Service (Amazon SES). Questo esempio utilizza un front-end creato con React.js per interagire con un backend Express Node.js.

- Integra un'applicazione web React.js con. Servizi AWS
- Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
- Invia un report per e-mail degli elementi di lavoro filtrati tramite Amazon SES.
- Distribuisci e gestisci risorse di esempio con lo AWS CloudFormation script incluso.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Aurora
- Amazon RDS
- Servizi di dati di Amazon RDS
- Amazon SES

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 azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`CreateCluster`

Il seguente esempio di codice mostra come usare. `CreateCluster`

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
import { 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 i dettagli sull'API, CreateClusterconsulta AWS SDK for JavaScript API Reference.

`DeleteCluster`

Il seguente esempio di codice mostra come utilizzare`DeleteCluster`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
import { 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 i dettagli sull'API, DeleteClusterconsulta AWS SDK for JavaScript API Reference.

`DescribeClusters`

Il seguente esempio di codice mostra come utilizzare`DescribeClusters`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
import { 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 };
```

Descrive i cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Per i dettagli sull'API, DescribeClustersconsulta AWS SDK for JavaScript API Reference.

`ModifyCluster`

Il seguente esempio di codice mostra come utilizzare`ModifyCluster`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
import { 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 i dettagli sull'API, ModifyClusterconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon Rekognition con SDK 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.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Scenari

Scenari

Creazione di un'applicazione serverless per gestire foto

L'esempio di codice seguente mostra come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per (v3) JavaScript

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API

- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Rilevamento di oggetti nelle immagini

L'esempio di codice seguente mostra come creare un'applicazione che utilizza Amazon Rekognition per rilevare oggetti in base a una categoria nelle immagini.

SDK per 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'applicazione invia all'amministratore una notifica e-mail sui risultati tramite Amazon Simple Email Service (Amazon SES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per rilevare gli oggetti tramite Amazon Rekognition.
- Verificare un indirizzo e-mail per Amazon SES.
- Inviare una notifica e-mail tramite Amazon SES.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Simple Storage Service (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.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Hello Amazon S3

Il seguente esempio di codice mostra come iniziare a usare Amazon S3.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  paginateListBuckets,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";
```

```
/**
 * List the S3 buckets in your configured AWS account.
 */
export const helloS3 = async () => {
  // When no region or credentials are provided, the SDK will use the
  // region and credentials from the local AWS config.
  const client = new S3Client({});

  try {
    /**
     * @type { import("@aws-sdk/client-s3").Bucket[] }
     */
    const buckets = [];

    for await (const page of paginateListBuckets({ client }, {})) {
      buckets.push(...page.Buckets);
    }
    console.log("Buckets: ");
    console.log(buckets.map((bucket) => bucket.Name).join("\n"));
    return buckets;
  } catch (caught) {
    // ListBuckets does not throw any modeled errors. Any error caught
    // here will be something generic like `AccessDenied`.
    if (caught instanceof S3ServiceException) {
      console.error(`${caught.name}: ${caught.message}`);
    } else {
      // Something besides S3 failed.
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, ListBucketsconsulta AWS SDK for JavaScript API Reference.

Nozioni di base

Informazioni 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.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Innanzitutto, importa tutti i moduli necessari.

```
// Used to check if currently running file is this file.
import { fileURLToPath } from "node:url";
import { readdirSync, readFileSync, writeFileSync } from "node: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, password } 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 {{ message: string }} options
   */
  password(options) {
    return password({ ...options, mask: true });
  }

  /**
   * @param {string} prompt
   */
  checkContinue = async (prompt = "") => {
    const prefix = prompt && `${prompt} `;
    const ok = await this.confirm({
      message: `${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 vengono 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 (const 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;
  }
  const copy = async () => {
    try {
      const sourceBucket = await prompter.input({
        message: "Enter source bucket name:",
```

```
      });
      const sourceKey = await prompter.input({
        message: "Enter source key:",
      });
      const destinationKey = await prompter.input({
        message: "Enter destination key:",
      });

      const command = new CopyObjectCommand({
        Bucket: destinationBucket,
        CopySource: `${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });
      await s3Client.send(command);
      await copyFileFromBucket({ destinationBucket });
    } catch (err) {
      console.error("Copy error.");
      console.error(err);
      const retryAnswer = await prompter.confirm({ message: "Try again?" });
      if (retryAnswer) {
        await copy();
      }
    }
  };
  await copy();
};
```

Non esiste un metodo SDK per ottenere più oggetti da un bucket. Creerai invece un elenco di oggetti da scaricare ed eseguirai iterazioni su di essi.

```
export const downloadFilesFromBucket = async ({ bucketName }) => {
  const { Contents } = await s3Client.send(
    new ListObjectsCommand({ Bucket: bucketName }),
  );
  const path = await prompter.input({
    message: "Enter destination path for files:",
  });

  for (const content of Contents) {
    const obj = await s3Client.send(
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
    );
```

```
    writeFileSync(
      `${path}/${content.Key}`,
      await obj.Body.transformToByteArray(),
    );
  }
  console.log("Files downloaded successfully.\n");
};
```

È il momento di eseguire una pulizia delle risorse. Prima di poter essere eliminato, un bucket deve essere vuoto. Queste due funzioni svuotano ed eliminano il bucket.

```
export const emptyBucket = async ({ bucketName }) => {
  const listObjectsCommand = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(listObjectsCommand);
  const keys = Contents.map((c) => c.Key);

  const deleteObjectsCommand = new DeleteObjectsCommand({
    Bucket: bucketName,
    Delete: { Objects: keys.map((key) => ({ Key: key })) },
  });
  await s3Client.send(deleteObjectsCommand);
  console.log(`${bucketName} emptied successfully.\n`);
};

export const deleteBucket = async ({ bucketName }) => {
  const command = new DeleteBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log(`${bucketName} deleted successfully.\n`);
};
```

La funzione "principale" esegue entrambe le operazioni. Se esegui direttamente questo file, verrà chiamata la funzione principale.

```
const main = async () => {
  const OBJECT_DIRECTORY = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../resources/sample_files/.sample_media`;

  try {
    console.log(wrapText("Welcome to the Amazon S3 getting started example."));
    console.log("Let's create a bucket.");
```

```
    const bucketName = await createBucket();
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("File upload."));
    console.log(
      "I have some default files ready to go. You can edit the source code to
 provide your own.",
    );
    await uploadFilesToBucket({
      bucketName,
      folderPath: OBJECT_DIRECTORY,
    });

    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Copy files."));
    await copyFileFromBucket({ destinationBucket: bucketName });
    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Download files."));
    await downloadFilesFromBucket({ bucketName });

    console.log(wrapText("Clean up."));
    await emptyBucket({ bucketName });
    await deleteBucket({ bucketName });
  } catch (err) {
    console.error(err);
  }
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2

- PutObject

Azioni

`CopyObject`

Il seguente esempio di codice mostra come usare`CopyObject`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Copia l'oggetto.

```
import {
  S3Client,
  CopyObjectCommand,
  ObjectNotInActiveTierError,
  waitUntilObjectExists,
} from "@aws-sdk/client-s3";

/**
 * Copy an S3 object from one bucket to another.
 *
 * @param {{
 *   sourceBucket: string,
 *   sourceKey: string,
 *   destinationBucket: string,
 *   destinationKey: string }} config
 */
export const main = async ({
  sourceBucket,
  sourceKey,
  destinationBucket,
  destinationKey,
}) => {
  const client = new S3Client({});
```

```
  try {
    await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucket}/${sourceKey}`,
        Bucket: destinationBucket,
        Key: destinationKey,
      }),
    );
    await waitUntilObjectExists(
      { client },
      { Bucket: destinationBucket, Key: destinationKey },
    );
    console.log(
      `Successfully copied ${sourceBucket}/${sourceKey} to ${destinationBucket}/
${destinationKey}`,
    );
  } catch (caught) {
    if (caught instanceof ObjectNotInActiveTierError) {
      console.error(
        `Could not copy ${sourceKey} from ${sourceBucket}. Object is not in the
 active tier.`,
      );
    } else {
      throw caught;
    }
  }
};
```

Copia l'oggetto a condizione ETag che non corrisponda a quello fornito.

```
import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
```

```
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 string, eTag: string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
  eTag,
}) => {
  const client = new S3Client({});
  const name = data.name;
  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucketName}/${sourceKeyName}`,
        Bucket: destinationBucketName,
        Key: `${name}${sourceKeyName}`,
        CopySourceIfMatch: eTag,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
 "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Unable to copy object "${sourceKeyName}" to bucket "${sourceBucketName}":
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
```

```
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Copia l'oggetto a condizione ETag che non corrisponda a quello fornito.

```
import {
```

```
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 string, eTag: string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
  eTag,
}) => {
  const client = new S3Client({});
  const name = data.name;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucketName}/${sourceKeyName}`,
        Bucket: destinationBucketName,
        Key: `${name}${sourceKeyName}`,
        CopySourceIfNoneMatch: eTag,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
 "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
```

```
        `Unable to copy object "${sourceKeyName}" to bucket "${sourceBucketName}":
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
```

```
  } else {
    console.error(errors.join("\n"));
  }
}
```

Copia l'oggetto a condizione che sia stato creato o modificato in un determinato periodo di tempo.

```
import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
}) => {
  const date = new Date();
  date.setDate(date.getDate() - 1);

  const name = data.name;
  const client = new S3Client({});
  const copySource = `${sourceBucketName}/${sourceKeyName}`;
  const copiedKey = name + sourceKeyName;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: copySource,
```

```
        Bucket: destinationBucketName,
        Key: copiedKey,
        CopySourceIfModifiedSince: date,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
 "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while copying object from ${sourceBucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
```

```
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Copia l'oggetto a condizione che non sia stato creato o modificato in un determinato periodo di tempo.

```
import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
}) => {
  const date = new Date();
```

```
  date.setDate(date.getDate() - 1);
  const client = new S3Client({});
  const name = data.name;
  const copiedKey = name + sourceKeyName;
  const copySource = `${sourceBucketName}/${sourceKeyName}`;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: copySource,
        Bucket: destinationBucketName,
        Key: copiedKey,
        CopySourceIfUnmodifiedSince: date,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
 "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while copying object from ${sourceBucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
```

```
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per i dettagli sull'API, consulta la CopyObjectsezione AWS SDK for JavaScript API Reference.

`CreateBucket`

Il seguente esempio di codice mostra come utilizzare`CreateBucket`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il bucket.

```
import {
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  CreateBucketCommand,
  S3Client,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

/**
 * Create an Amazon S3 bucket.
 * @param {{ bucketName: string }} config
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { Location } = await client.send(
      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: bucketName,
      }),
    );
    await waitUntilBucketExists({ client }, { Bucket: bucketName });
    console.log(`Bucket created with location ${Location}`);
  } catch (caught) {
    if (caught instanceof BucketAlreadyExists) {
      console.error(
        `The bucket "${bucketName}" already exists in another AWS account. Bucket
 names must be globally unique.`,
      );
    }
    // WARNING: If you try to create a bucket in the North Virginia region,
    // and you already own a bucket in that region with the same name, this
    // error will not be thrown. Instead, the call will return successfully
    // and the ACL on that bucket will be reset.
    else if (caught instanceof BucketAlreadyOwnedByYou) {
      console.error(
        `The bucket "${bucketName}" already exists in this AWS account.`,
      );
    } else {
```

```
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateBucketconsulta AWS SDK for JavaScript API Reference.

`DeleteBucket`

Il seguente esempio di codice mostra come utilizzare`DeleteBucket`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina il bucket.

```
import {
  DeleteBucketCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Delete an Amazon S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new DeleteBucketCommand({
    Bucket: bucketName,
  });

  try {
    await client.send(command);
    console.log("Bucket was deleted.");
```

```
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting bucket. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting the bucket. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteBucketconsulta AWS SDK for JavaScript API Reference.

`DeleteBucketPolicy`

Il seguente esempio di codice mostra come utilizzare`DeleteBucketPolicy`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina la policy del bucket.

```
import {
  DeleteBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";
```

```
/**
 * Remove the policy from an Amazon S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteBucketPolicyCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`Bucket policy deleted from "${bucketName}".`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting policy from ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting policy from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteBucketPolicyconsulta AWS SDK for JavaScript API Reference.

`DeleteBucketWebsite`

Il seguente esempio di codice mostra come utilizzare`DeleteBucketWebsite`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina la configurazione del sito web dal bucket.

```
import {
  DeleteBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Remove the website configuration for a bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteBucketWebsiteCommand({
        Bucket: bucketName,
      }),
    );
    // The response code will be successful for both removed configurations and
    // configurations that did not exist in the first place.
    console.log(
      `The bucket "${bucketName}" is not longer configured as a website, or it never
 was.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while removing website configuration from ${bucketName}. The
 bucket doesn't exist.`,
```

```
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while removing website configuration from ${bucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteBucketWebsiteconsulta AWS SDK for JavaScript API Reference.

`DeleteObject`

Il seguente esempio di codice mostra come utilizzare`DeleteObject`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina un oggetto.

```
import {
  DeleteObjectCommand,
  S3Client,
  S3ServiceException,
  waitUntilObjectNotExists,
} from "@aws-sdk/client-s3";

/**
 * Delete one object from an Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
```

```
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteObjectCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    await waitUntilObjectNotExists(
      { client },
      { Bucket: bucketName, Key: key },
    );
    // A successful delete, or a delete for a non-existent object, both return
    // a 204 response code.
    console.log(
      `The object "${key}" from bucket "${bucketName}" was deleted, or it didn't
 exist.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting object from ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, DeleteObjectconsulta AWS SDK for JavaScript API Reference.

`DeleteObjects`

Il seguente esempio di codice mostra come utilizzare`DeleteObjects`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina più oggetti.

```
import {
  DeleteObjectsCommand,
  S3Client,
  S3ServiceException,
  waitUntilObjectNotExists,
} from "@aws-sdk/client-s3";

/**
 * Delete multiple objects from an S3 bucket.
 * @param {{ bucketName: string, keys: string[] }}
 */
export const main = async ({ bucketName, keys }) => {
  const client = new S3Client({});

  try {
    const { Deleted } = await client.send(
      new DeleteObjectsCommand({
        Bucket: bucketName,
        Delete: {
          Objects: keys.map((k) => ({ Key: k })),
        },
      }),
    );
    for (const key in keys) {
      await waitUntilObjectNotExists(
        { client },
        { Bucket: bucketName, Key: key },
      );
    }
```

```
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted
 objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting objects from ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting objects from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, DeleteObjectsconsulta AWS SDK for JavaScript API Reference.

`GetBucketAcl`

Il seguente esempio di codice mostra come utilizzare`GetBucketAcl`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera le autorizzazioni ACL.

```
import {
  GetBucketAclCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Retrieves the Access Control List (ACL) for an S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetBucketAclCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`ACL for bucket "${bucketName}":`);
    console.log(JSON.stringify(response, null, 2));
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting ACL for ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting ACL for ${bucketName}. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per i dettagli sull'API, GetBucketAclconsulta AWS SDK for JavaScript API Reference.

`GetBucketCors`

Il seguente esempio di codice mostra come utilizzare`GetBucketCors`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la policy CORS per il bucket.

```
import {
  GetBucketCorsCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the Cross-Origin Resource Sharing (CORS) configuration information
 * set for the bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new GetBucketCorsCommand({
    Bucket: bucketName,
  });

  try {
    const { CORSRules } = await client.send(command);
    console.log(JSON.stringify(CORSRules));
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `\n${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders}`,
        `\nAllowedMethods: ${cr.AllowedMethods}`,
```

```
        `\nAllowedOrigins: ${cr.AllowedOrigins}`,
        `\nExposeHeaders: ${cr.ExposeHeaders}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting bucket CORS rules for ${bucketName}. The bucket
 doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting bucket CORS rules for ${bucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, GetBucketCorsconsulta AWS SDK for JavaScript API Reference.

`GetBucketPolicy`

Il seguente esempio di codice mostra come utilizzare`GetBucketPolicy`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la policy del bucket.

```
import {
  GetBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Logs the policy for a specified bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { Policy } = await client.send(
      new GetBucketPolicyCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`Policy for "${bucketName}":\n${Policy}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting policy from ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting policy from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, GetBucketPolicyconsulta AWS SDK for JavaScript API Reference.

`GetBucketWebsite`

Il seguente esempio di codice mostra come utilizzare`GetBucketWebsite`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Recupera la configurazione del sito web.

```
import {
  GetBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the website configuration for a bucket.
 * @param {{ bucketName }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetBucketWebsiteCommand({
        Bucket: bucketName,
      }),
    );
    console.log(
      `Your bucket is set up to host a website with the following configuration:\n
${JSON.stringify(response, null, 2)}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchWebsiteConfiguration"
    ) {
      console.error(
```

```
        `Error from S3 while getting website configuration for ${bucketName}. The
 bucket isn't configured as a website.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting website configuration for ${bucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, GetBucketWebsiteconsulta AWS SDK for JavaScript API Reference.

`GetObject`

Il seguente esempio di codice mostra come utilizzare`GetObject`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Scarica l'oggetto.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
```

```
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

Scarica l'oggetto a condizione che ETag corrisponda a quello fornito.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
```

```
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
```

```
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Scarica l'oggetto a condizione ETag che non corrisponda a quello fornito.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
```

```
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfNoneMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
```

```
    key: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Scarica l'oggetto a condizione che sia stato creato o modificato in un determinato periodo di tempo.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const date = new Date();
  date.setDate(date.getDate() - 1);
```

```
  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfModifiedSince: date,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
```

```
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Scarica l'oggetto a condizione che non sia stato creato o modificato in un determinato periodo di tempo.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const date = new Date();
  date.setDate(date.getDate() - 1);
  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfUnmodifiedSince: date,
```

```
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
```

```
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta la GetObjectsezione AWS SDK for JavaScript API Reference.

`GetObjectLegalHold`

Il seguente esempio di codice mostra come utilizzare`GetObjectLegalHold`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  GetObjectLegalHoldCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get an object's current legal hold status.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
```

```
  try {
    const response = await client.send(
      new GetObjectLegalHoldCommand({
        Bucket: bucketName,
        Key: key,
        // Optionally, you can provide additional parameters
        // ExpectedBucketOwner: "<account ID that is expected to own the bucket>",
        // VersionId: "<the specific version id of the object to check>",
      }),
    );
    console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting legal hold status for ${key} in ${bucketName}.
 The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting legal hold status for ${key} in ${bucketName}
 from ${bucketName}.  ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
```

```
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per i dettagli sull'API, GetObjectLegalHoldconsulta AWS SDK for JavaScript API Reference.

`GetObjectLockConfiguration`

Il seguente esempio di codice mostra come utilizzare`GetObjectLockConfiguration`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  GetObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Gets the Object Lock configuration for a bucket.
```

```
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { ObjectLockConfiguration } = await client.send(
      new GetObjectLockConfigurationCommand({
        Bucket: bucketName,
        // Optionally, you can provide additional parameters
        // ExpectedBucketOwner: "<account ID that is expected to own the bucket>",
      }),
    );
    console.log(
      `Object Lock Configuration:\n${JSON.stringify(ObjectLockConfiguration)}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting object lock configuration for ${bucketName}.
 The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object lock configuration for ${bucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
```

```
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per i dettagli sull'API, GetObjectLockConfigurationconsulta AWS SDK for JavaScript API Reference.

`GetObjectRetention`

Il seguente esempio di codice mostra come utilizzare`GetObjectRetention`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  GetObjectRetentionCommand,
  S3Client,
  S3ServiceException,
```

```
} from "@aws-sdk/client-s3";

/**
 * Log the "RetainUntilDate" for an object in an S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const { Retention } = await client.send(
      new GetObjectRetentionCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    console.log(
      `${key} in ${bucketName} will be retained until ${Retention.RetainUntilDate}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchObjectLockConfiguration"
    ) {
      console.warn(
        `The object "${key}" in the bucket "${bucketName}" does not have an
 ObjectLock configuration.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object retention settings for "${bucketName}".
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
```

```
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per i dettagli sull'API, GetObjectRetentionconsulta AWS SDK for JavaScript API Reference.

`ListBuckets`

Il seguente esempio di codice mostra come utilizzare`ListBuckets`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i bucket.

```
import {
  paginateListBuckets,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * List the Amazon S3 buckets in your account.
 */
export const main = async () => {
  const client = new S3Client({});
  /** @type {?import('@aws-sdk/client-s3').Owner} */
  let Owner = null;

  /** @type {import('@aws-sdk/client-s3').Bucket[]} */
  const Buckets = [];

  try {
    const paginator = paginateListBuckets({ client }, {});

    for await (const page of paginator) {
      if (!Owner) {
        Owner = page.Owner;
      }

      Buckets.push(...page.Buckets);
    }

    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`,
    );
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (caught) {
    if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while listing buckets.  ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
```

```
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListBucketsconsulta AWS SDK for JavaScript API Reference.

`ListObjectsV2`

Il seguente esempio di codice mostra come utilizzare`ListObjectsV2`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca tutti gli oggetti nel bucket. Se è presente più di un oggetto, IsTruncated NextContinuationToken verrà utilizzato per scorrere l'elenco completo.

```
import {
  S3Client,
  S3ServiceException,
  // This command supersedes the ListObjectsCommand and is the recommended way to
 list objects.
  paginateListObjectsV2,
} from "@aws-sdk/client-s3";

/**
 * Log all of the object keys in a bucket.
 * @param {{ bucketName: string, pageSize: string }}
 */
export const main = async ({ bucketName, pageSize }) => {
  const client = new S3Client({});
  /** @type {string[][]} */
  const objects = [];
  try {
    const paginator = paginateListObjectsV2(
      { client, /* Max items per page */ pageSize: Number.parseInt(pageSize) },
```

```
      { Bucket: bucketName },
    );

    for await (const page of paginator) {
      objects.push(page.Contents.map((o) => o.Key));
    }
    objects.forEach((objectList, pageNum) => {
      console.log(
        `Page ${pageNum + 1}\n------\n${objectList.map((o) => `•
 ${o}`).join("\n")}\n`,
      );
    });
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while listing objects for "${bucketName}". The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while listing objects for "${bucketName}".  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, consulta la versione ListObjectsV2 in AWS SDK for JavaScript API Reference.

`PutBucketAcl`

Il seguente esempio di codice mostra come utilizzare`PutBucketAcl`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inserisci l'ACL del bucket.

```
import {
  PutBucketAclCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Grant read access to a user using their canonical AWS account ID.
 *
 * 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.
 * @param {{ bucketName: string, granteeCanonicalUserId: string,
 ownerCanonicalUserId }}
 */
export const main = async ({
  bucketName,
  granteeCanonicalUserId,
  ownerCanonicalUserId,
}) => {
  const client = new S3Client({});
  const command = new PutBucketAclCommand({
    Bucket: bucketName,
    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: granteeCanonicalUserId,
            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: "READ",
        },
      ],
      Owner: {
        ID: ownerCanonicalUserId,
      },
    },
  });

  try {
    await client.send(command);
    console.log(`Granted READ access to ${bucketName}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while setting ACL for bucket ${bucketName}. The bucket
 doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting ACL for bucket ${bucketName}. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutBucketAclconsulta AWS SDK for JavaScript API Reference.

`PutBucketCors`

Il seguente esempio di codice mostra come utilizzare`PutBucketCors`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiungi una regola CORS.

```
import {
  PutBucketCorsCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Allows cross-origin requests to an S3 bucket by setting the CORS configuration.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new PutBucketCorsCommand({
        Bucket: bucketName,
        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,
            },
          ],
        },
      }),
    );
    console.log(`Successfully set CORS rules for bucket: ${bucketName}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while setting CORS rules for ${bucketName}. The bucket
 doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting CORS rules for ${bucketName}. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutBucketCorsconsulta AWS SDK for JavaScript API Reference.

`PutBucketPolicy`

Il seguente esempio di codice mostra come utilizzare`PutBucketPolicy`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiungi la policy.

```
import {
  PutBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Grant an IAM role GetObject access to all of the objects
 * in the provided bucket.
 * @param {{ bucketName: string, iamRoleArn: string }}
 */
export const main = async ({ bucketName, iamRoleArn }) => {
  const client = new S3Client({});
  const command = new PutBucketPolicyCommand({
    // This is a resource-based policy. For more information on resource-based
 policies,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/
access_policies.html#policies_resource-based.
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {
            AWS: iamRoleArn,
          },
          Action: "s3:GetObject",
          Resource: `arn:aws:s3:::${bucketName}/*`,
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: bucketName,
```

```
  });

  try {
    await client.send(command);
    console.log(
      `GetObject access to the bucket "${bucketName}" was granted to the provided
 IAM role.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "MalformedPolicy"
    ) {
      console.error(
        `Error from S3 while setting the bucket policy for the bucket
 "${bucketName}". The policy was malformed.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting the bucket policy for the bucket
 "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutBucketPolicyconsulta AWS SDK for JavaScript API Reference.

`PutBucketWebsite`

Il seguente esempio di codice mostra come utilizzare`PutBucketWebsite`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Imposta la configurazione del sito web.

```
import {
  PutBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Configure an Amazon S3 bucket to serve a static website.
 * Website access must also be granted separately. For more information
 * on setting the permissions for website access, see
 * https://docs.aws.amazon.com/AmazonS3/latest/userguide/
WebsiteAccessPermissionsReqd.html.
 *
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new PutBucketWebsiteCommand({
    Bucket: bucketName,
    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 when the request is
        // for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
```

```
    await client.send(command);
    console.log(
      `The bucket "${bucketName}" has been configured as a static website.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while configuring the bucket "${bucketName}" as a static
 website. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while configuring the bucket "${bucketName}" as a static
 website. ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, PutBucketWebsiteconsulta AWS SDK for JavaScript API Reference.

`PutObject`

Il seguente esempio di codice mostra come utilizzare`PutObject`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Carica l'oggetto.

```
import { readFile } from "node:fs/promises";

import {
  PutObjectCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Upload a file to an S3 bucket.
 * @param {{ bucketName: string, key: string, filePath: string }}
 */
export const main = async ({ bucketName, key, filePath }) => {
  const client = new S3Client({});
  const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: key,
    Body: await readFile(filePath),
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "EntityTooLarge"
    ) {
      console.error(
        `Error from S3 while uploading object to ${bucketName}. \
The object was too large. To upload objects larger than 5GB, use the S3 console
 (160GB max) \
or the multipart upload API (5TB max).`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while uploading object to ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
```

```
};
```

Carica l'oggetto a condizione che ETag corrisponda a quello fornito.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
```

```
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per i dettagli sull'API, consulta la PutObjectsezione AWS SDK for JavaScript API Reference.

`PutObjectLegalHold`

Il seguente esempio di codice mostra come utilizzare`PutObjectLegalHold`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  PutObjectLegalHoldCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Apply a legal hold configuration to the specified object.
 * @param {{ bucketName: string, objectKey: string, legalHoldStatus: "ON" | "OFF" }}
 */
export const main = async ({ bucketName, objectKey, legalHoldStatus }) => {
  if (!["OFF", "ON"].includes(legalHoldStatus.toUpperCase())) {
    throw new Error(
      "Invalid parameter. legalHoldStatus must be 'ON' or 'OFF'.",
    );
  }

  const client = new S3Client({});
  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: legalHoldStatus,
    },
  });
```

```
  try {
    await client.send(command);
    console.log(
      `Legal hold status set to "${legalHoldStatus}" for "${objectKey}" in
 "${bucketName}"`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying legal hold status for "${objectKey}" in
 "${bucketName}". The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying legal hold status for "${objectKey}" in
 "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    objectKey: {
      type: "string",
      required: true,
    },
    legalHoldStatus: {
```

```
      type: "string",
      default: "ON",
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per i dettagli sull'API, PutObjectLegalHoldconsulta AWS SDK for JavaScript API Reference.

`PutObjectLockConfiguration`

Il seguente esempio di codice mostra come utilizzare`PutObjectLockConfiguration`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Imposta la configurazione Object Lock di un bucket.

```
import {
  PutObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
```

```
 * Enable S3 Object Lock for an Amazon S3 bucket.
 * After you enable Object Lock on a bucket, you can't
 * disable Object Lock or suspend versioning for that bucket.
 * @param {{ bucketName: string, enabled: boolean }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
  });

  try {
    await client.send(command);
    console.log(`Object Lock for "${bucketName}" enabled.`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying the object lock configuration for the bucket
 "${bucketName}". The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying the object lock configuration for the bucket
 "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";
```

```
const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Imposta il periodo di conservazione predefinito di un bucket.

```
import {
  PutObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Change the default retention settings for an object in an Amazon S3 bucket.
 * @param {{ bucketName: string, retentionDays: string }}
 */
export const main = async ({ bucketName, retentionDays }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: bucketName,
```

```
      // The Object Lock configuration that you want to apply to the specified
bucket.
      ObjectLockConfiguration: {
        ObjectLockEnabled: "Enabled",
        Rule: {
          // The default Object Lock retention mode and period that you want to
apply
          // to new objects placed in the specified bucket. Bucket settings
require
          // both a mode and a period. The period can be either Days or Years but
          // you must select one.
          DefaultRetention: {
            // In governance mode, users can't overwrite or delete an object
version
            // or alter its lock settings unless they have special permissions.
With
            // governance mode, you protect objects against being deleted by most
users,
            // but you can still grant some users permission to alter the
retention settings
            // or delete the objects if necessary.
            Mode: "GOVERNANCE",
            Days: Number.parseInt(retentionDays),
          },
        },
      },
    }),
  );
  console.log(
    `Set default retention mode to "GOVERNANCE" with a retention period of
${retentionDays} day(s).`,
  );
 } catch (caught) {
  if (
    caught instanceof S3ServiceException &&
    caught.name === "NoSuchBucket"
  ) {
    console.error(
      `Error from S3 while setting the default object retention for a bucket. The
bucket doesn't exist.`,
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
```

```
        `Error from S3 while setting the default object retention for a bucket.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    retentionDays: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per i dettagli sull'API, PutObjectLockConfigurationconsulta AWS SDK for JavaScript API Reference.

`PutObjectRetention`

Il seguente esempio di codice mostra come utilizzare`PutObjectRetention`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  PutObjectRetentionCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Place a 24-hour retention period on an object in an Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: key,
    BypassGovernanceRetention: false,
    Retention: {
      // In governance mode, users can't overwrite or delete an object version
      // or alter its lock settings unless they have special permissions. With
      // governance mode, you protect objects against being deleted by most users,
      // but you can still grant some users permission to alter the retention
 settings
      // or delete the objects if necessary.
      Mode: "GOVERNANCE",
      RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
  });
```

```
  try {
    await client.send(command);
    console.log("Object Retention settings updated.");
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying the governance mode and retention period on
 an object. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying the governance mode and retention period on
 an object. ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
```

```
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Per i dettagli sull'API, PutObjectRetentionconsulta AWS SDK for JavaScript API Reference.

Scenari

Creazione di un URL prefirmato

L'esempio di codice seguente mostra come creare un URL prefirmato per Amazon S3 e caricare un oggetto.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un URL prefirmato per caricare un oggetto in un bucket.

```
import https from "node:https";

import { XMLParser } from "fast-xml-parser";
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 });
};

/**
 * Make a PUT request to the provided URL.
 *
 * @param {string} url
 * @param {string} data
 */
const 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", () => {
          const parser = new XMLParser();
          if (res.statusCode >= 200 && res.statusCode <= 299) {
            resolve(parser.parse(responseBody, true));
```

```
          } else {
            reject(parser.parse(responseBody, true));
          }
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
};

/**
 * Create two presigned urls for uploading an object to an S3 bucket.
 * The first presigned URL is created with credentials from the shared INI file
 * in the current environment. The second presigned URL is created using an
 * existing S3Client instance that has already been provided with credentials.
 * @param {{ bucketName: string, key: string, region: string }}
 */
export const main = async ({ bucketName, key, region }) => {
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      bucket: bucketName,
      key,
      region,
    });

    const clientUrl = await createPresignedUrlWithClient({
      bucket: bucketName,
      region,
      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 (caught) {
    if (caught instanceof Error && caught.name === "CredentialsProviderError") {
      console.error(
        `There was an error getting your credentials. Are your local credentials
 configured?\n${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

Crea un URL prefirmato per scaricare un oggetto da un bucket.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(new HttpRequest(url));
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new GetObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};
```

```
/**
 * Create two presigned urls for downloading an object from an S3 bucket.
 * The first presigned URL is created with credentials from the shared INI file
 * in the current environment. The second presigned URL is created using an
 * existing S3Client instance that has already been provided with credentials.
 * @param {{ bucketName: string, key: string, region: string }}
 */
export const main = async ({ bucketName, key, region }) => {
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      bucket: bucketName,
      region,
      key,
    });

    const clientUrl = await createPresignedUrlWithClient({
      bucket: bucketName,
      region,
      key,
    });

    console.log("Presigned URL without client");
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "CredentialsProviderError") {
      console.error(
        `There was an error getting your credentials. Are your local credentials
 configured?\n${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

Creazione di un'applicazione serverless per gestire foto

L'esempio di codice seguente mostra come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per JavaScript (v3)

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (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.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Il codice seguente è il componente React pertinente che effettua chiamate all' AWS SDK. Una versione eseguibile dell'applicazione contenente questo componente è disponibile al link precedente. GitHub

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  type ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";

function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
 role to the pool,
      // and grant the role access to the 's3:GetObject' action.
      //
      // You'll also need to configure the CORS settings on the bucket to allow
 traffic from
      // this example site. Here's an example configuration that allows all origins.
 Don't
      // do this in production.
      //[
      //  {
      //    "AllowedHeaders": ["*"],
      //    "AllowedMethods": ["GET"],
      //    "AllowedOrigins": ["*"],
      //    "ExposeHeaders": [],
      //  },
      //]
      //
      credentials: fromCognitoIdentityPool({
        clientConfig: { region: "us-east-1" },
        identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
      }),
    });
    const command = new ListObjectsCommand({ Bucket: "bucket-name" });
    client.send(command).then(({ Contents }) => setObjects(Contents || []));
```

```
  }, []);

  return (
    <div className="App">
      {objects.map((o) => (
        <div key={o.ETag}>{o.Key}</div>
      ))}
    </div>
  );
}

export default App;
```

- Per i dettagli sull'API, consulta la sezione API ListObjectsReference AWS SDK for JavaScript .

Creazione di un'applicazione Amazon Textract explorer

L'esempio di codice seguente mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDK per JavaScript (v3)

Mostra come utilizzare per AWS SDK for JavaScript creare un'applicazione React che utilizza Amazon Textract per estrarre dati dall'immagine di un documento e visualizzarli in una pagina Web interattiva. Questo esempio viene eseguito in un browser Web e richiede, come credenziali, un'identità autenticata Amazon Cognito. Utilizza Amazon Simple Storage Service (Amazon S3) per l'archiviazione e per le notifiche esegue il polling di una coda Amazon Simple Queue Service (Amazon SQS) sottoscritta a un argomento 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
- Simple Storage Service (Amazon S3)
- Amazon SNS
- Amazon SQS
- Amazon Textract

Eliminare tutti gli oggetti in un bucket

L'esempio di codice seguente mostra come eliminare più oggetti da un bucket Amazon S3.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina tutti gli oggetti in un bucket Amazon S3 specificato.

```
import {
  DeleteObjectsCommand,
  paginateListObjectsV2,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 *
 * @param {{ bucketName: string }} config
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  try {
    console.log(`Deleting all objects in bucket: ${bucketName}`);

    const paginator = paginateListObjectsV2(
      { client },
      {
        Bucket: bucketName,
      },
    );

    const objectKeys = [];
    for await (const { Contents } of paginator) {
      objectKeys.push(...Contents.map((obj) => ({ Key: obj.Key })));
    }

    const deleteCommand = new DeleteObjectsCommand({
```

```
      Bucket: bucketName,
      Delete: { Objects: objectKeys },
    });

    await client.send(deleteCommand);

    console.log(`All objects deleted from bucket: ${bucketName}`);
  } catch (caught) {
    if (caught instanceof Error) {
      console.error(
        `Failed to empty ${bucketName}. ${caught.name}: ${caught.message}`,
      );
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    bucketName: {
      type: "string",
    },
  };

  const { values } = parseArgs({ options });
  main(values);
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - DeleteObjects
  - ListObjectsV2

Rilevamento di oggetti nelle immagini

L'esempio di codice seguente mostra come creare un'applicazione che utilizza Amazon Rekognition per rilevare oggetti in base a una categoria nelle immagini.

SDK per 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'applicazione invia all'amministratore una notifica e-mail sui risultati tramite Amazon Simple Email Service (Amazon SES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per rilevare gli oggetti tramite Amazon Rekognition.
- Verificare un indirizzo e-mail per Amazon SES.
- Inviare una notifica e-mail tramite Amazon SES.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SES

Bloccare gli oggetti di Amazon S3

L'esempio di codice seguente mostra come utilizzare le funzionalità Object Lock di S3.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Punto di ingresso per lo scenario (index.js). In questo modo vengono orchestrate tutte le fasi. 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,
  getBucketPrefix,
  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"),
        getBucketPrefix(scenarios),
        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 "node: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, {
    name: "Amazon S3 object locking workflow",
    description:
      "Work with Amazon Simple Storage Service (Amazon S3) object locking
 features.",
    synopsis:
      "node index.js --scenario <deploy | demo | clean> [-h|--help] [-y|--yes] [-
v|--verbose]",
  });
}
```

Restituisce i messaggi di benvenuto alla console (welcome.steps.js).

```
/**
 * @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 Feature
 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 };
```

Distribuisce bucket, oggetti e impostazioni dei file (setup.steps.js).

```
import {
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,
  GetBucketVersioningCommand,
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  S3ServiceException,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */
```

```
/**
 * @param {Scenarios} scenarios
 */
const getBucketPrefix = (scenarios) =>
  new scenarios.ScenarioInput(
    "bucketPrefix",
    "Provide a prefix that will be used for bucket creation.",
    { type: "input", default: "amzn-s3-demo-bucket" },
  );

/**
 * @param {Scenarios} scenarios
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    (state) => `The following buckets will be created:
         ${state.bucketPrefix}-no-lock with object lock False.
         ${state.bucketPrefix}-lock-enabled with object lock True.
         ${state.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 = `${state.bucketPrefix}-no-lock`;
    const lockEnabledBucketName = `${state.bucketPrefix}-lock-enabled`;
    const retentionBucketName = `${state.bucketPrefix}-retention-after-creation`;

    try {
      await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
      await waitUntilBucketExists({ client }, { Bucket: noLockBucketName });
```

```
      await client.send(
        new CreateBucketCommand({
          Bucket: lockEnabledBucketName,
          ObjectLockEnabledForBucket: true,
        }),
      );
      await waitUntilBucketExists(
        { client },
        { Bucket: lockEnabledBucketName },
      );
      await client.send(
        new CreateBucketCommand({ Bucket: retentionBucketName }),
      );
      await waitUntilBucketExists({ client }, { Bucket: retentionBucketName });

      state.noLockBucketName = noLockBucketName;
      state.lockEnabledBucketName = lockEnabledBucketName;
      state.retentionBucketName = retentionBucketName;
    } catch (caught) {
      if (
        caught instanceof BucketAlreadyExists ||
        caught instanceof BucketAlreadyOwnedByYou
      ) {
        console.error(`${caught.name}: ${caught.message}`);
        state.earlyExit = true;
      } else {
        throw caught;
      }
    }
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    (state) => `The following test files will be created:
         file0.txt in ${state.bucketPrefix}-no-lock.
         file1.txt in ${state.bucketPrefix}-no-lock.
         file0.txt in ${state.bucketPrefix}-lock-enabled.
         file1.txt in ${state.bucketPrefix}-lock-enabled.
         file0.txt in ${state.bucketPrefix}-retention-after-creation.
         file1.txt in ${state.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) => {
    try {
      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,
        }),
      );
    } catch (caught) {
      if (caught instanceof S3ServiceException) {
        console.error(
          `Error from S3 while uploading object.  ${caught.name}:
 ${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  });

/**
 * @param {Scenarios} scenarios
 */
const updateRetention = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateRetention",
    (state) => `A bucket can be configured to use object locking with a default
 retention period.
```

```
A default retention period will be configured for ${state.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,
        },
      }),
    );

    const getBucketVersioning = new GetBucketVersioningCommand({
      Bucket: state.retentionBucketName,
    });

    await retry({ intervalInMs: 500, maxRetries: 10 }, async () => {
      const { Status } = await client.send(getBucketVersioning);
      if (Status !== "Enabled") {
        throw new Error("Bucket versioning is not 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",
    (state) => `Object lock policies can also be added to existing buckets.
An object lock policy will be added to ${state.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 {
```

```
  getBucketPrefix,
  createBuckets,
  confirmCreateBuckets,
  createBucketsAction,
  populateBuckets,
  confirmPopulateBuckets,
  populateBucketsAction,
  updateRetention,
  confirmUpdateRetention,
  updateRetentionAction,
  updateLockPolicy,
  confirmUpdateLockPolicy,
  updateLockPolicyAction,
  confirmSetLegalHoldFileEnabled,
  setLegalHoldFileEnabledAction,
  confirmSetRetentionPeriodFileEnabled,
  setRetentionPeriodFileEnabledAction,
  confirmSetLegalHoldFileRetention,
  setLegalHoldFileRetentionAction,
  confirmSetRetentionPeriodFileRetention,
  setRetentionPeriodFileRetentionAction,
};
```

Visualizza ed elimina i file nei bucket (repl.steps.js).

```
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 };
```

Elimina tutte le risorse create (clean.steps.js).

```
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;
        }
        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 informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold
  - PutObjectLockConfiguration
  - PutObjectRetention

Eseguire richieste condizionali

L'esempio di codice seguente mostra come aggiungere precondizioni alle richieste Amazon S3.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Punto di ingresso per il flusso di lavoro (index.js). In questo modo vengono orchestrate tutte le fasi. 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,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
} 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 Conditional Requests - Deploy",
      [
        welcome(scenarios),
        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        getBucketPrefix(scenarios),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
        createBucketsAction(scenarios, client),
        populateBuckets(scenarios),
        confirmPopulateBuckets(scenarios),
```

```
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        saveState,
      ],
      initialState,
    ),
    demo: new scenarios.Scenario(
      "S3 Conditional Requests - Demo",
      [loadState, welcome(scenarios), replAction(scenarios, client)],
      initialState,
    ),
    clean: new scenarios.Scenario(
      "S3 Conditional Requests - Destroy",
      [
        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
      ],
      initialState,
    ),
  };
};

// Call function if run directly
import { fileURLToPath } from "node: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, {
    name: "Amazon S3 object locking workflow",
    description:
      "Work with Amazon Simple Storage Service (Amazon S3) object locking
 features.",
    synopsis:
      "node index.js --scenario <deploy | demo | clean> [-h|--help] [-y|--yes] [-
v|--verbose]",
  });
}
```

Restituisce i messaggi di benvenuto alla console (welcome.steps.js).

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    "This example demonstrates the use of conditional requests for S3 operations." +
      " You can use conditional requests to add preconditions to S3 read requests to
 return " +
      "or copy an object based on its Entity tag (ETag), or last modified date.You
 can use " +
      "a conditional write requests to prevent overwrites by ensuring there is no
 existing " +
      "object with the same key.\n" +
      "This example will enable you to perform conditional reads and writes that
 will succeed " +
      "or fail based on your selected options.\n" +
      "Sample buckets and a sample object will be created as part of the example.\n"
 +
      "Some steps require a key name prefix to be defined by the user. Before you
 begin, you can " +
      "optionally edit this prefix in ./object_name.json. If you do so, please
 reload the scenario before you begin.",
    { 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 };
```

Distribuisce bucket e oggetti (setup.steps.js).

```
import {
  ChecksumAlgorithm,
  CreateBucketCommand,
  PutObjectCommand,
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  S3ServiceException,
  waitUntilBucketExists,
} 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 getBucketPrefix = (scenarios) =>
  new scenarios.ScenarioInput(
    "bucketPrefix",
    "Provide a prefix that will be used for bucket creation.",
    { type: "input", default: "amzn-s3-demo-bucket" },
  );
/**
 * @param {Scenarios} scenarios
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    (state) => `The following buckets will be created:
         ${state.bucketPrefix}-source-bucket.
         ${state.bucketPrefix}-destination-bucket.`,
    { 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 sourceBucketName = `${state.bucketPrefix}-source-bucket`;
    const destinationBucketName = `${state.bucketPrefix}-destination-bucket`;

    try {
      await client.send(
        new CreateBucketCommand({
          Bucket: sourceBucketName,
        }),
      );
      await waitUntilBucketExists({ client }, { Bucket: sourceBucketName });
      await client.send(
        new CreateBucketCommand({
          Bucket: destinationBucketName,
        }),
      );
      await waitUntilBucketExists(
        { client },
        { Bucket: destinationBucketName },
      );

      state.sourceBucketName = sourceBucketName;
      state.destinationBucketName = destinationBucketName;
    } catch (caught) {
      if (
        caught instanceof BucketAlreadyExists ||
        caught instanceof BucketAlreadyOwnedByYou
      ) {
        console.error(`${caught.name}: ${caught.message}`);
        state.earlyExit = true;
      } else {
```

```
        throw caught;
      }
    }
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    (state) => `The following test files will be created:
         file01.txt in ${state.bucketPrefix}-source-bucket.`,
    { 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) => {
    try {
      await client.send(
        new PutObjectCommand({
          Bucket: state.sourceBucketName,
          Key: "file01.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
    } catch (caught) {
      if (caught instanceof S3ServiceException) {
        console.error(
```

```
          `Error from S3 while uploading object.  ${caught.name}:
 ${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  });

export {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
};
```

Ottiene, copia e inserisce oggetti utilizzando le richieste condizionali S3 (repl.steps.js).

```
import path from "node:path";
import { fileURLToPath } from "node:url";
import { dirname } from "node:path";

import {
  ListObjectVersionsCommand,
  GetObjectCommand,
  CopyObjectCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";
import data from "./object_name.json" assert { type: "json" };
import { readFile } from "node:fs/promises";
import {
  ScenarioInput,
  Scenario,
  ScenarioAction,
  ScenarioOutput,
} from "../../../libs/scenario/index.js";

/**
```

```
 * @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,
  CONDITIONAL_READ: 2,
  CONDITIONAL_COPY: 3,
  CONDITIONAL_WRITE: 4,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new ScenarioInput(
    "replChoice",
    "Explore the S3 conditional request features by selecting one of the following
 choices",
    {
      type: "select",
      choices: [
        { name: "Print list of bucket items.", value: choices.LIST_ALL_FILES },
        {
          name: "Perform a conditional read.",
          value: choices.CONDITIONAL_READ,
        },
        {
          name: "Perform a conditional copy. These examples use the key name prefix
 defined in ./object_name.json.",
          value: choices.CONDITIONAL_COPY,
        },
        {
          name: "Perform a conditional write. This example use the sample file ./
text02.txt.",
          value: choices.CONDITIONAL_WRITE,
        },
        { 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 } = version;
      files.push({ bucket, key: Key });
    }
  }
  return files;
};

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 * @param {string} key
 */
const getEtag = async (client, bucket, key) => {
  const objectsResponse = await client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    }),
  );
  return objectsResponse.ETag;
};

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
```

```
 */
export const replAction = (scenarios, client) =>
  new ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.sourceBucketName,
        state.destinationBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file to use:",
        {
          type: "select",
          choices: files.map((file, index) => ({
            name: `${index + 1}: ${file.bucket}: ${file.key} (Etag: ${
              file.version
            })`,
            value: index,
          })),
        },
      );
      const condReadOptions = new scenarios.ScenarioInput(
        "selectOption",
        "Which conditional read action would you like to take?",
        {
          type: "select",
          choices: [
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",
            "If-Modified-Since: using yesterday's date. This condition should
 succeed.",
            "If-Unmodified-Since: using yesterday's date. This condition should
 fail.",
          ],
        },
      );
      const condCopyOptions = new scenarios.ScenarioInput(
        "selectOption",
        "Which conditional copy action would you like to take?",
        {
          type: "select",
          choices: [
```

```
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",
            "If-Modified-Since: using yesterday's date. This condition should
succeed.",
            "If-Unmodified-Since: using yesterday's date. This condition should
fail.",
          ],
        },
      );
      const condWriteOptions = new scenarios.ScenarioInput(
        "selectOption",
        "Which conditional write action would you like to take?",
        {
          type: "select",
          choices: [
            "IfNoneMatch condition on the object key: If the key is a duplicate, the
write will fail.",
          ],
        },
      );

      const { replChoice } = state;

      switch (replChoice) {
        case choices.LIST_ALL_FILES: {
          const files = await getAllFiles(client, [
            state.sourceBucketName,
            state.destinationBucketName,
          ]);
          state.replOutput = files
            .map(
              (file) => `Items in bucket ${file.bucket}: object: ${file.key} `,
            )
            .join("\n");
          break;
        }
        case choices.CONDITIONAL_READ:
          {
            const selectedCondRead = await condReadOptions.handle(state);
            if (
              selectedCondRead ===
              "If-Match: using the object's ETag. This condition should succeed."
            ) {
              const bucket = state.sourceBucketName;
```

```
              const key = "file01.txt";
              const ETag = await getEtag(client, bucket, key);

              try {
                await client.send(
                  new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfMatch: ETag,
                  }),
                );
                state.replOutput = `${key} in bucket ${state.sourceBucketName} read
because ETag provided matches the object's ETag.`;
              } catch (err) {
                state.replOutput = `Unable to read object ${key} in bucket
${state.sourceBucketName}: ${err.message}`;
              }
              break;
            }
            if (
              selectedCondRead ===
              "If-None-Match: using the object's ETag. This condition should fail."
            ) {
              const bucket = state.sourceBucketName;
              const key = "file01.txt";
              const ETag = await getEtag(client, bucket, key);

              try {
                await client.send(
                  new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfNoneMatch: ETag,
                  }),
                );
                state.replOutput = `${key} in ${state.sourceBucketName} was
returned.`;
              } catch (err) {
                state.replOutput = `${key} in ${state.sourceBucketName} was not
read: ${err.message}`;
              }
              break;
            }
            if (
```

```
              selectedCondRead ===
              "If-Modified-Since: using yesterday's date. This condition should
succeed."
            ) {
              const date = new Date();
              date.setDate(date.getDate() - 1);

              const bucket = state.sourceBucketName;
              const key = "file01.txt";
              try {
                await client.send(
                  new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfModifiedSince: date,
                  }),
                );
                state.replOutput = `${key} in bucket ${state.sourceBucketName} read
because it has been created or modified in the last 24 hours.`;
              } catch (err) {
                state.replOutput = `Unable to read object ${key} in bucket
${state.sourceBucketName}: ${err.message}`;
              }
              break;
            }
            if (
              selectedCondRead ===
              "If-Unmodified-Since: using yesterday's date. This condition should
fail."
            ) {
              const bucket = state.sourceBucketName;
              const key = "file01.txt";

              const date = new Date();
              date.setDate(date.getDate() - 1);
              try {
                await client.send(
                  new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfUnmodifiedSince: date,
                  }),
                );
                state.replOutput = `${key} in ${state.sourceBucketName} was read.`;
```

```
            } catch (err) {
              state.replOutput = `${key} in ${state.sourceBucketName} was not
read: ${err.message}`;
            }
            break;
          }
        }
        break;
      case choices.CONDITIONAL_COPY: {
        const selectedCondCopy = await condCopyOptions.handle(state);
        if (
          selectedCondCopy ===
          "If-Match: using the object's ETag. This condition should succeed."
        ) {
          const bucket = state.sourceBucketName;
          const key = "file01.txt";
          const ETag = await getEtag(client, bucket, key);

          const copySource = `${bucket}/${key}`;
          // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
          const name = data.name;
          const copiedKey = `${name}${key}`;
          try {
            await client.send(
              new CopyObjectCommand({
                CopySource: copySource,
                Bucket: state.destinationBucketName,
                Key: copiedKey,
                CopySourceIfMatch: ETag,
              }),
            );
            state.replOutput = `${key} copied as ${copiedKey} to bucket
${state.destinationBucketName} because ETag provided matches the object's ETag.`;
          } catch (err) {
            state.replOutput = `Unable to copy object ${key} as ${copiedKey} to
bucket ${state.destinationBucketName}: ${err.message}`;
          }
          break;
        }
        if (
          selectedCondCopy ===
          "If-None-Match: using the object's ETag. This condition should fail."
        ) {
```

```
            const bucket = state.sourceBucketName;
            const key = "file01.txt";
            const ETag = await getEtag(client, bucket, key);
            const copySource = `${bucket}/${key}`;
            // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
            const name = data.name;
            const copiedKey = `${name}${key}`;

            try {
              await client.send(
                new CopyObjectCommand({
                  CopySource: copySource,
                  Bucket: state.destinationBucketName,
                  Key: copiedKey,
                  CopySourceIfNoneMatch: ETag,
                }),
              );
              state.replOutput = `${copiedKey} copied to bucket
${state.destinationBucketName}`;
            } catch (err) {
              state.replOutput = `Unable to copy object as ${key} as as ${copiedKey}
to bucket ${state.destinationBucketName}: ${err.message}`;
            }
            break;
          }
          if (
            selectedCondCopy ===
            "If-Modified-Since: using yesterday's date. This condition should
succeed."
          ) {
            const bucket = state.sourceBucketName;
            const key = "file01.txt";
            const copySource = `${bucket}/${key}`;
            // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
            const name = data.name;
            const copiedKey = `${name}${key}`;

            const date = new Date();
            date.setDate(date.getDate() - 1);

            try {
              await client.send(
```

```
              new CopyObjectCommand({
                CopySource: copySource,
                Bucket: state.destinationBucketName,
                Key: copiedKey,
                CopySourceIfModifiedSince: date,
              }),
            );
            state.replOutput = `${key} copied as ${copiedKey} to bucket
${state.destinationBucketName} because it has been created or modified in the last
24 hours.`;
          } catch (err) {
            state.replOutput = `Unable to copy object ${key} as ${copiedKey} to
bucket ${state.destinationBucketName} : ${err.message}`;
          }
          break;
        }
        if (
          selectedCondCopy ===
          "If-Unmodified-Since: using yesterday's date. This condition should
fail."
        ) {
          const bucket = state.sourceBucketName;
          const key = "file01.txt";
          const copySource = `${bucket}/${key}`;
          // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
          const name = data.name;
          const copiedKey = `${name}${key}`;

          const date = new Date();
          date.setDate(date.getDate() - 1);

          try {
            await client.send(
              new CopyObjectCommand({
                CopySource: copySource,
                Bucket: state.destinationBucketName,
                Key: copiedKey,
                CopySourceIfUnmodifiedSince: date,
              }),
            );
            state.replOutput = `${copiedKey} copied to bucket
${state.destinationBucketName} because it has not been created or modified in the
last 24 hours.`;
```

```
            } catch (err) {
              state.replOutput = `Unable to copy object ${key} to bucket
${state.destinationBucketName}: ${err.message}`;
            }
          }
          break;
        }
        case choices.CONDITIONAL_WRITE:
          {
            const selectedCondWrite = await condWriteOptions.handle(state);
            if (
              selectedCondWrite ===
              "IfNoneMatch condition on the object key: If the key is a duplicate,
the write will fail."
            ) {
              // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
              const key = "text02.txt";
              const __filename = fileURLToPath(import.meta.url);
              const __dirname = dirname(__filename);
              const filePath = path.join(__dirname, "text02.txt");
              try {
                await client.send(
                  new PutObjectCommand({
                    Bucket: `${state.destinationBucketName}`,
                    Key: `${key}`,
                    Body: await readFile(filePath),
                    IfNoneMatch: "*",
                  }),
                );
                state.replOutput = `${key} uploaded to bucket
${state.destinationBucketName} because the key is not a duplicate.`;
              } catch (err) {
                state.replOutput = `Unable to upload object to bucket
${state.destinationBucketName}:${err.message}`;
              }
              break;
            }
          }
          break;

        default:
          throw new Error(`Invalid replChoice: ${replChoice}`);
      }
```

```
    },
    {
      whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new ScenarioOutput("REPL output", (state) => state.replOutput, {
          preformatted: true,
        }),
      },
    },
  );

export { replInput, choices };
```

Elimina tutte le risorse create (clean.steps.js).

```
import {
  DeleteObjectCommand,
  DeleteBucketCommand,
  ListObjectVersionsCommand,
} 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 { sourceBucketName, destinationBucketName } = state;
    const buckets = [sourceBucketName, destinationBucketName].filter((b) => b);

    for (const bucket of buckets) {
      try {
        let objectsResponse;
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
        );
        for (const version of objectsResponse.Versions || []) {
          const { Key, VersionId } = version;
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: bucket,
                Key,
                VersionId,
              }),
            );
          } catch (err) {
            console.log(`An error occurred: ${err.message} `);
          }
        }
      } catch (e) {
        if (e instanceof Error && e.name === "NoSuchBucket") {
          console.log("Objects and buckets have already been deleted.");
          continue;
        }
        throw e;
      }

      await client.send(new DeleteBucketCommand({ Bucket: bucket }));
      console.log(`Delete for ${bucket} complete.`);
    }
  });

export { confirmCleanup, cleanupAction };
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - CopyObject
  - GetObject
  - PutObject

Caricamento o download di file di grandi dimensioni

L'esempio di codice seguente mostra come caricare o scaricare file di grandi dimensioni in e da Amazon S3.

Per ulteriori informazioni, consulta Caricamento di un oggetto utilizzando il caricamento in più parti.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Carica un file di grandi dimensioni.

```
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

import {
  ProgressBar,
  logger,
} from "@aws-doc-sdk-examples/lib/utils/util-log.js";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

/**
 * Create a 25MB file and upload it in parts to the specified
 * Amazon S3 bucket.
```

```
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const str = createString();
  const buffer = Buffer.from(str, "utf8");
  const progressBar = new ProgressBar({
    description: `Uploading "${key}" to "${bucketName}"`,
    barLength: 30,
  });

  try {
    const upload = new Upload({
      client: new S3Client({}),
      params: {
        Bucket: bucketName,
        Key: key,
        Body: buffer,
      },
    });

    upload.on("httpUploadProgress", ({ loaded, total }) => {
      progressBar.update({ current: loaded, total });
    });

    await upload.done();
  } catch (caught) {
    if (caught instanceof Error && caught.name === "AbortError") {
      logger.error(`Multipart upload was aborted. ${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

Scarica un file di grandi dimensioni.

```
import { fileURLToPath } from "node:url";
import { GetObjectCommand, NoSuchKey, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream, rmSync } from "node: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: Number.parseInt(start),
    end: Number.parseInt(end),
    length: Number.parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

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 };

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
    });
    console.log(`Downloaded bytes ${nextRange.start} to ${nextRange.end}`);
```

```
    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

/**
 * Download a large object from and Amazon S3 bucket.
 *
 * 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.
 *
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  try {
    await downloadInChunks({
      bucket: bucketName,
      key: key,
    });
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(`Failed to download object. No such key "${key}".`);
      rmSync(key);
    }
  }
};
```

Esempi serverless

Invocazione di una funzione Lambda da un trigger Amazon S3

L'esempio di codice seguente 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 evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. 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);
  }
};
```

SageMaker Esempi di intelligenza artificiale che utilizzano SDK for (v3 JavaScript)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con SageMaker AI.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Ciao SageMaker AI

Il seguente esempio di codice mostra come iniziare a usare l' SageMaker IA.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  SageMakerClient,
  ListNotebookInstancesCommand,
} from "@aws-sdk/client-sagemaker";

const client = new SageMakerClient({
  region: "us-west-2",
});

export const helloSagemaker = async () => {
  const command = new ListNotebookInstancesCommand({ MaxResults: 5 });

  const response = await client.send(command);
  console.log(
```

```
    "Hello Amazon SageMaker! Let's list some of your notebook instances:",
  );

  const instances = response.NotebookInstances || [];

  if (instances.length === 0) {
    console.log(
      "• No notebook instances found. Try creating one in the AWS Management Console
 or with the CreateNotebookInstanceCommand.",
    );
  } else {
    console.log(
      instances
        .map(
          (i) =>
            `• Instance: ${i.NotebookInstanceName}\n  Arn:${
              i.NotebookInstanceArn
            } \n  Creation Date: ${i.CreationTime.toISOString()}`,
        )
        .join("\n"),
    );
  }

  return response;
};
```

- Per i dettagli sull'API, ListNotebookInstancesconsulta AWS SDK for JavaScript API Reference.

Azioni

`CreatePipeline`

Il seguente esempio di codice mostra come utilizzare`CreatePipeline`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Una funzione che crea una pipeline SageMaker AI utilizzando una definizione JSON fornita localmente.

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../scenarios/features/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 i dettagli sull'API, consulta la sezione AWS SDK for JavaScript API CreatePipelineReference.

`DeletePipeline`

Il seguente esempio di codice mostra come utilizzare`DeletePipeline`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

La sintassi per eliminare una pipeline SageMaker AI. Questo codice fa parte di una funzione più ampia. Fai riferimento a «Crea una pipeline» o al repository per ulteriori informazioni. GitHub

```
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
```

- Per i dettagli sull'API, consulta DeletePipelinela sezione API Reference.AWS SDK for JavaScript

`DescribePipelineExecution`

Il seguente esempio di codice mostra come utilizzare`DescribePipelineExecution`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Attendi che l'esecuzione di una pipeline SageMaker AI 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;
  const intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error("Pipeline was forcefully stopped.");
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- Per i dettagli sull'API, consulta la sezione DescribePipelineExecution AWS SDK for JavaScriptAPI Reference.

`StartPipelineExecution`

Il seguente esempio di codice mostra come utilizzare`StartPipelineExecution`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Avvia l'esecuzione di una pipeline SageMaker AI.

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };
```

```
  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
```

```
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}
```

- Per i dettagli sull'API, consulta la sezione StartPipelineExecution AWS SDK for JavaScriptAPI Reference.

Scenari

Nozioni di base su processi geospaziali e pipeline

L'esempio di codice seguente mostra come:

- Configurare le risorse per una pipeline.
- Configurare una pipeline che esegua un processo geospaziale.
- Avvio dell'esecuzione di una pipeline.
- Monitorare lo stato dell'esecuzione.
- Visualizzare l'output della pipeline.
- Eliminare le risorse.

Per ulteriori informazioni, consulta Creare ed eseguire SageMaker pipeline utilizzando AWS SDKs Community.aws.

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Il seguente estratto di file contiene funzioni che utilizzano il client SageMaker AI per gestire una pipeline.

```
import { readFileSync } from "node: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;
      }
      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,
  )}../../../../../scenarios/features/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,
    )}../../../../../scenarios/features/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;
  const 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 pipeline SageMaker AI, 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 informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution
  - UpdatePipeline

Esempi di Secrets Manager con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) with Secrets Manager.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`GetSecretValue`

Il seguente esempio di codice mostra come utilizzare`GetSecretValue`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";

export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: secretName,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
  //   CreatedDate: 2023-08-08T19:29:51.294Z,
  //   Name: 'binary-secret-3873048',
  //   SecretBinary: Uint8Array(11) [
  //      98, 105, 110, 97, 114,
  //     121,  32, 100, 97, 116,
  //      97
  //   ],
  //   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
  //   VersionStages: [ 'AWSCURRENT' ]
  // }

  if (response.SecretString) {
    return response.SecretString;
  }

  if (response.SecretBinary) {
    return response.SecretBinary;
  }
};
```

- Per i dettagli sull'API, GetSecretValueconsulta AWS SDK for JavaScript API Reference.

Esempi di Amazon SES con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon SES.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni
- Scenari

Azioni

`CreateReceiptFilter`

Il seguente esempio di codice mostra come usare`CreateReceiptFilter`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  CreateReceiptFilterCommand,
```

```
  ReceiptFilterPolicy,
} from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
 address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
 addressesOptions.
      },
      /*
        The name of the IP address filter. Only ASCII letters, numbers, underscores,
 or dashes.
        Must be less than 64 characters and start and end with a letter or number.
       */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });

  try {
    return await sesClient.send(createReceiptFilterCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Per i dettagli sull'API, CreateReceiptFilterconsulta AWS SDK for JavaScript API Reference.

`CreateReceiptRule`

Il seguente esempio di codice mostra come utilizzare`CreateReceiptRule`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateReceiptRuleCommand, TlsPolicy } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
```

```
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
  });
};

const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};
```

- Per i dettagli sull'API, CreateReceiptRuleconsulta AWS SDK for JavaScript API Reference.

`CreateReceiptRuleSet`

Il seguente esempio di codice mostra come utilizzare`CreateReceiptRuleSet`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateReceiptRuleSetCommand } from "@aws-sdk/client-ses";
```

```
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createCreateReceiptRuleSetCommand = (ruleSetName) => {
  return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
};

const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- Per i dettagli sull'API, CreateReceiptRuleSetconsulta AWS SDK for JavaScript API Reference.

`CreateTemplate`

Il seguente esempio di codice mostra come utilizzare`CreateTemplate`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
```

```
const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template
 system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, CreateTemplateconsulta AWS SDK for JavaScript API Reference.

`DeleteIdentity`

Il seguente esempio di codice mostra come utilizzare`DeleteIdentity`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, DeleteIdentityconsulta AWS SDK for JavaScript API Reference.

`DeleteReceiptFilter`

Il seguente esempio di codice mostra come utilizzare`DeleteReceiptFilter`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteReceiptFilterCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, DeleteReceiptFilterconsulta AWS SDK for JavaScript API Reference.

`DeleteReceiptRule`

Il seguente esempio di codice mostra come utilizzare`DeleteReceiptRule`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteReceiptRuleCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleCommand = () => {
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, DeleteReceiptRuleconsulta AWS SDK for JavaScript API Reference.

`DeleteReceiptRuleSet`

Il seguente esempio di codice mostra come utilizzare`DeleteReceiptRuleSet`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};

const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, DeleteReceiptRuleSetconsulta AWS SDK for JavaScript API Reference.

`DeleteTemplate`

Il seguente esempio di codice mostra come utilizzare`DeleteTemplate`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, DeleteTemplateconsulta AWS SDK for JavaScript API Reference.

`GetTemplate`

Il seguente esempio di codice mostra come utilizzare`GetTemplate`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Per i dettagli sull'API, GetTemplateconsulta AWS SDK for JavaScript API Reference.

`ListIdentities`

Il seguente esempio di codice mostra come utilizzare`ListIdentities`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, ListIdentitiesconsulta AWS SDK for JavaScript API Reference.

`ListReceiptFilters`

Il seguente esempio di codice mostra come utilizzare`ListReceiptFilters`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { ListReceiptFiltersCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
};
```

- Per i dettagli sull'API, ListReceiptFiltersconsulta AWS SDK for JavaScript API Reference.

`ListTemplates`

Il seguente esempio di codice mostra come utilizzare`ListTemplates`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
```

```
};
```

- Per i dettagli sull'API, ListTemplatesconsulta AWS SDK for JavaScript API Reference.

`SendBulkTemplatedEmail`

Il seguente esempio di codice mostra come utilizzare`SendBulkTemplatedEmail`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
```

```
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map
 each user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
```

```
};
```

- Per i dettagli sull'API, SendBulkTemplatedEmailconsulta AWS SDK for JavaScript API Reference.

`SendEmail`

Il seguente esempio di codice mostra come utilizzare`SendEmail`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
```

```
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Per i dettagli sull'API, SendEmailconsulta AWS SDK for JavaScript API Reference.

`SendRawEmail`

Il seguente esempio di codice mostra come utilizzare`SendRawEmail`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Usare nodemailer per inviare un'e-mail con un allegato.

```
import sesClientModule from "@aws-sdk/client-ses";
/**
 * nodemailer wraps the SES SDK and calls SendRawEmail. Use this for more advanced
 * functionality like adding attachments to your email.
 *
 * https://nodemailer.com/transports/ses
 */
import nodemailer from "nodemailer";

/**
 * @param {string} from An Amazon SES verified email address.
 * @param {*} to An Amazon SES verified email address.
 */
export const sendEmailWithAttachments = (
  from = "from@example.com",
  to = "to@example.com",
) => {
  const ses = new sesClientModule.SESClient({});
  const transporter = nodemailer.createTransport({
    SES: { ses, aws: sesClientModule },
  });

  return new Promise((resolve, reject) => {
    transporter.sendMail(
      {
        from,
        to,
        subject: "Hello World",
        text: "Greetings from Amazon SES!",
        attachments: [{ content: "Hello World!", filename: "hello.txt" }],
      },
      (err, info) => {
        if (err) {
```

```
          reject(err);
        } else {
          resolve(info);
        }
      },
    );
  });
};
```

- Per i dettagli sull'API, SendRawEmailconsulta AWS SDK for JavaScript API Reference.

`SendTemplatedEmail`

Il seguente esempio di codice mostra come utilizzare`SendTemplatedEmail`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");
```

```
const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
 party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Per i dettagli sull'API, SendTemplatedEmailconsulta AWS SDK for JavaScript API Reference.

`UpdateTemplate`

Il seguente esempio di codice mostra come utilizzare`UpdateTemplate`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, UpdateTemplateconsulta AWS SDK for JavaScript API Reference.

`VerifyDomainIdentity`

Il seguente esempio di codice mostra come utilizzare`VerifyDomainIdentity`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, VerifyDomainIdentityconsulta AWS SDK for JavaScript API Reference.

`VerifyEmailIdentity`

Il seguente esempio di codice mostra come utilizzare`VerifyEmailIdentity`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

- Per i dettagli sull'API, VerifyEmailIdentityconsulta AWS SDK for JavaScript API Reference.

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.

SDK per JavaScript (v3)

Mostra come utilizzare Amazon Transcribe per creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia i risultati per e-mail tramite Amazon Simple Email Service (Amazon SES).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

L'esempio di codice seguente mostra come creare un'applicazione web che traccia gli elementi di lavoro in database Amazon Aurora serverless e utilizza Amazon Simple Email Service (Amazon SES) per inviare report.

SDK per JavaScript (v3)

Mostra come utilizzare AWS SDK for JavaScript (v3) per creare un'applicazione Web che tenga traccia degli elementi di lavoro in un database Amazon Aurora e invii report tramite e-mail utilizzando Amazon Simple Email Service (Amazon SES). Questo esempio utilizza un front-end creato con React.js per interagire con un backend Express Node.js.

- Integra un'applicazione web React.js con. Servizi AWS
- Elenca, aggiungi e aggiorna elementi in una tabella Aurora.
- Invia un report per e-mail degli elementi di lavoro filtrati tramite Amazon SES.

- Distribuisci e gestisci risorse di esempio con lo AWS CloudFormation script incluso.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Aurora
- Amazon RDS
- Servizi di dati di Amazon RDS
- Amazon SES

Rilevamento di oggetti nelle immagini

L'esempio di codice seguente mostra come creare un'applicazione che utilizza Amazon Rekognition per rilevare oggetti in base a una categoria nelle immagini.

SDK per 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'applicazione invia all'amministratore una notifica e-mail sui risultati tramite Amazon Simple Email Service (Amazon SES).

Scopri come:

- Creare un utente non autenticato tramite Amazon Cognito.
- Analizza le immagini per rilevare gli oggetti tramite Amazon Rekognition.
- Verificare un indirizzo e-mail per Amazon SES.
- Inviare una notifica e-mail tramite Amazon SES.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SES

Esempi di Amazon SNS con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon SNS.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Hello Amazon SNS

Il seguente esempio di codice mostra come iniziare a usare Amazon SNS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inizializza un client SNS ed elenca gli argomenti nel tuo account.

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";
```

```
export const helloSns = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
  const topics = [];

  for await (const page of paginatedTopics) {
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your account.`,
  );
  console.log(topics.map((t) => `  * ${t.TopicArn}`).join("\n"));
};
```

- Per i dettagli sull'API, ListTopicsconsulta AWS SDK for JavaScript API Reference.

Azioni

`CheckIfPhoneNumberIsOptedOut`

Il seguente esempio di codice mostra come utilizzare`CheckIfPhoneNumberIsOptedOut`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.

- Per i dettagli sull'API, CheckIfPhoneNumberIsOptedOutconsulta AWS SDK for JavaScript API Reference.

`ConfirmSubscription`

Il seguente esempio di codice mostra come utilizzare`ConfirmSubscription`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
```

```
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ConfirmSubscriptionconsulta AWS SDK for JavaScript API Reference.

`CreateTopic`

Il seguente esempio di codice mostra come utilizzare`CreateTopic`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateTopicconsulta AWS SDK for JavaScript API Reference.

`DeleteTopic`

Il seguente esempio di codice mostra come utilizzare`DeleteTopic`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteTopicconsulta AWS SDK for JavaScript API Reference.

`GetSMSAttributes`

Il seguente esempio di codice mostra come utilizzare`GetSMSAttributes`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
```

```
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta Get SMSAttributes in AWS SDK for JavaScript API Reference.

`GetTopicAttributes`

Il seguente esempio di codice mostra come usare`GetTopicAttributes`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, GetTopicAttributesconsulta AWS SDK for JavaScript API Reference.

`ListSubscriptions`

Il seguente esempio di codice mostra come utilizzare`ListSubscriptions`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
```

```
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListSubscriptionsconsulta AWS SDK for JavaScript API Reference.

`ListTopics`

Il seguente esempio di codice mostra come utilizzare`ListTopics`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListTopicsconsulta AWS SDK for JavaScript API Reference.

`Publish`

Il seguente esempio di codice mostra come utilizzare`Publish`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                  if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
```

```
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

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;
    let deduplicationId;
    let 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 informazioni dettagliate sull'API, consulta Pubblicazione nella documentazione di riferimento dell'API AWS SDK for JavaScript .

`SetSMSAttributes`

Il seguente esempio di codice mostra come usare`SetSMSAttributes`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
```

```
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, consulta Set SMSAttributes in AWS SDK for JavaScript API Reference.

`SetTopicAttributes`

Il seguente esempio di codice mostra come utilizzare`SetTopicAttributes`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, SetTopicAttributesconsulta AWS SDK for JavaScript API Reference.

`Subscribe`

Il seguente esempio di codice mostra come utilizzare`Subscribe`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
```

```
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

Sottoscrive 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;
};
```

Sottoscrive 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;
};
```

Sottoscrive una coda SQS 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;
};
```

Sottoscrive 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 informazioni dettagliate sull'API, consulta Sottoscrizione nella documentazione di riferimento dell'API AWS SDK for JavaScript .

`Unsubscribe`

Il seguente esempio di codice mostra come usare`Unsubscribe`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare il client in un modulo separato ed esportarlo.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importare l'SDK e i moduli client e chiamare l'API.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
```

```
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per informazioni dettagliate sull'API, consulta Annullamento della sottoscrizione nella documentazione di riferimento dell'API AWS SDK for JavaScript .

Scenari

Costruisci un'app per inviare dati a una tabella DynamoDB

L'esempio di codice seguente mostra come costruire un'applicazione che invia dati a una tabella Amazon DynamoDB e che ti avvisa quando un utente aggiorna la tabella.

SDK per JavaScript (v3)

Questo esempio mostra come creare un'app che consenta agli utenti di inviare dati a una tabella Amazon DynamoDB e un messaggio di testo all'amministratore utilizzando Amazon Simple Notification Service (Amazon SNS).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SNS

Creazione di un'applicazione serverless per gestire foto

L'esempio di codice seguente mostra come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per JavaScript (v3)

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Creazione di un'applicazione Amazon Textract explorer

L'esempio di codice seguente mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDK per JavaScript (v3)

Mostra come utilizzare per AWS SDK for JavaScript creare un'applicazione React che utilizza Amazon Textract per estrarre dati dall'immagine di un documento e visualizzarli in una pagina Web interattiva. Questo esempio viene eseguito in un browser Web e richiede, come credenziali, un'identità autenticata Amazon Cognito. Utilizza Amazon Simple Storage Service (Amazon S3) per l'archiviazione e per le notifiche esegue il polling di una coda Amazon Simple Queue Service (Amazon SQS) sottoscritta a un argomento 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
- Simple Storage Service (Amazon S3)

- Amazon SNS
- Amazon SQS
- Amazon Textract

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Creazione di un argomento (FIFO o non FIFO).
- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo è il punto di ingresso per questo scenario.

```
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";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Il codice precedente fornisce le dipendenze necessarie e avvia lo scenario. 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.
    const 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;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
```

```
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
```

```
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
```

```
    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
```

```
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Pubblicare
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Utilizzo di Gateway API per invocare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da Amazon API Gateway.

SDK per JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando l'API di JavaScript runtime Lambda. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo

esempio viene illustrato come creare una funzione Lambda invocata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

Utilizzo degli eventi pianificati per invocare una funzione Lambda

Il seguente esempio di codice mostra come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

SDK per JavaScript (v3)

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3.

Servizi utilizzati in questo esempio

- CloudWatch Registri

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Esempi serverless

Invocazione di una funzione Lambda da un trigger Amazon SNS

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dal ricevimento di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento SNS con JavaScript Lambda utilizzando.

```
// 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;
  }
}
```

Consumo di un evento SNS con TypeScript Lambda utilizzando.

```
// 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;
  }
}
```

Esempi di Amazon SQS con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon SQS.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Hello Amazon SQS

Il seguente esempio di codice mostra come iniziare a usare Amazon SQS.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inizializza un client Amazon SQS ed elenca le code.

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
```

```
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- Per i dettagli sull'API, ListQueuesconsulta AWS SDK for JavaScript API Reference.

Azioni

`ChangeMessageVisibility`

Il seguente esempio di codice mostra come utilizzare`ChangeMessageVisibility`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Riceve un messaggio Amazon SQS e ne modifica la visibilità del timeout.

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
```

```
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 1,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- Per i dettagli sull'API, ChangeMessageVisibilityconsulta AWS SDK for JavaScript API Reference.

`CreateQueue`

Il seguente esempio di codice mostra come utilizzare`CreateQueue`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea una coda standard Amazon SQS.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Crea una coda Amazon SQS con polling lungo.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: {
        // When the wait time for the ReceiveMessage API action is greater than 0,
        // long polling is in effect. The maximum long polling wait time is 20
        // seconds. Long polling helps reduce the cost of using Amazon SQS by,
        // eliminating the number of empty responses and false empty responses.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        ReceiveMessageWaitTimeSeconds: "20",
      },
    }),
```

```
  );
  console.log(response);
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, CreateQueueconsulta AWS SDK for JavaScript API Reference.

`DeleteMessage`

Il seguente esempio di codice mostra come utilizzare`DeleteMessage`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Riceve ed elimina messaggi Amazon SQS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
```

```
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Per i dettagli sull'API, DeleteMessageconsulta AWS SDK for JavaScript API Reference.

`DeleteMessageBatch`

Il seguente esempio di codice mostra come utilizzare`DeleteMessageBatch`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
```

```
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Per i dettagli sull'API, DeleteMessageBatchconsulta AWS SDK for JavaScript API Reference.

`DeleteQueue`

Il seguente esempio di codice mostra come utilizzare`DeleteQueue`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina una coda Amazon SQS.

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, DeleteQueueconsulta AWS SDK for JavaScript API Reference.

`GetQueueAttributes`

Il seguente esempio di codice mostra come utilizzare`GetQueueAttributes`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- Per i dettagli sull'API, GetQueueAttributesconsulta AWS SDK for JavaScript API Reference.

`GetQueueUrl`

Il seguente esempio di codice mostra come utilizzare`GetQueueUrl`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottiene l'URL di una coda Amazon SQS.

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, GetQueueUrlconsulta AWS SDK for JavaScript API Reference.

`ListQueues`

Il seguente esempio di codice mostra come utilizzare`ListQueues`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca le code Amazon SQS.

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    for (const url of urls) {
      console.log(url);
    }
  }

  return urls;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, ListQueuesconsulta AWS SDK for JavaScript API Reference.

`ReceiveMessage`

Il seguente esempio di codice mostra come utilizzare`ReceiveMessage`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Riceve un messaggio da una coda Amazon SQS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
```

```
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

Riceve un messaggio da una coda Amazon SQS utilizzando il supporto del long polling.

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
    // The duration (in seconds) for which the call waits for a message
    // to arrive in the queue before returning. If a message is available,
    // the call returns sooner than WaitTimeSeconds. If no messages are
    // available and the wait time expires, the call returns successfully
    // with an empty list of messages.
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
    WaitTimeSeconds: 20,
  });

  const response = await client.send(command);
  console.log(response);
```

```
  return response;
};
```

- Per i dettagli sull'API, ReceiveMessageconsulta AWS SDK for JavaScript API Reference.

`SendMessage`

Il seguente esempio di codice mostra come utilizzare`SendMessage`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invia un messaggio a una coda Amazon SQS.

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
```

```
    },
    MessageBody:
      "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK for JavaScript.
- Per i dettagli sull'API, SendMessageconsulta AWS SDK for JavaScript API Reference.

`SetQueueAttributes`

Il seguente esempio di codice mostra come utilizzare`SetQueueAttributes`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

  const response = await client.send(command);
```

```
  console.log(response);
  return response;
};
```

Configura una coda Amazon SQS per utilizzare il long polling.

```
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 DLQ.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
```

```
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Per i dettagli sull'API, SetQueueAttributesconsulta AWS SDK for JavaScript API Reference.

Scenari

Creazione di un'applicazione Amazon Textract explorer

L'esempio di codice seguente mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDK per JavaScript (v3)

Mostra come utilizzare per AWS SDK for JavaScript creare un'applicazione React che utilizza Amazon Textract per estrarre dati dall'immagine di un documento e visualizzarli in una pagina Web interattiva. Questo esempio viene eseguito in un browser Web e richiede, come credenziali, un'identità autenticata Amazon Cognito. Utilizza Amazon Simple Storage Service (Amazon S3) per l'archiviazione e per le notifiche esegue il polling di una coda Amazon Simple Queue Service (Amazon SQS) sottoscritta a un argomento 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
- Simple Storage Service (Amazon S3)

- Amazon SNS
- Amazon SQS
- Amazon Textract

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Creazione di un argomento (FIFO o non FIFO).
- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo è il punto di ingresso per questo scenario.

```
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";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Il codice precedente fornisce le dipendenze necessarie e avvia lo scenario. 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.
    const 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;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
```

```
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
```

```
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
```

```
    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
```

```
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Pubblicare
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Esempi serverless

Invocazione di una funzione Lambda da un trigger Amazon SQS

L'esempio di codice seguente 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.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento SQS con JavaScript Lambda utilizzando.

```
// 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;
  }
}
```

Consumo di un evento SQS con TypeScript Lambda utilizzando.

```
// 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 Amazon SQS

L'esempio di codice seguente mostra come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. 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 SQS utilizzando Lambda. JavaScript

```
// 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 batch SQS utilizzando Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

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 azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Azioni

`StartExecution`

Il seguente esempio di codice mostra come utilizzare`StartExecution`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { 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 "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- Per i dettagli sull'API, StartExecutionconsulta AWS SDK for JavaScript API Reference.

AWS STS esempi che utilizzano SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con. AWS STS

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`AssumeRole`

Il seguente esempio di codice mostra come utilizzare`AssumeRole`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Assumi il ruolo IAM.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
```

```
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, AssumeRoleconsulta AWS SDK for JavaScript API Reference.

Supporto 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. Supporto

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni

Nozioni di base

Salve Supporto

L'esempio di codice seguente mostra come iniziare a utilizzare Supporto.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invocazione di `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.",
      );
    }
    throw err;
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- Per i dettagli sull'API, DescribeServicesconsulta AWS SDK for JavaScript API Reference.

Nozioni di base

Informazioni 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.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo nel terminale.

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
```

```
} from "@aws-sdk/client-support";
import * 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.",
      );
    }
    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 informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases
  - DescribeCommunications
  - DescribeServices
  - DescribeSeverityLevels
  - ResolveCase

Azioni

`AddAttachmentsToSet`

Il seguente esempio di codice mostra come usare`AddAttachmentsToSet`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { AddAttachmentsToSetCommand } from "@aws-sdk/client-support";
```

```
import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new attachment set or add attachments to an existing set.
    // Provide an 'attachmentSetId' value to add attachments to an existing set.
    // Use AddCommunicationToCase or CreateCase to associate an attachment set with
 a support case.
    const response = await client.send(
      new AddAttachmentsToSetCommand({
        // You can add up to three attachments per set. The size limit is 5 MB per
 attachment.
        attachments: [
          {
            fileName: "example.txt",
            data: new TextEncoder().encode("some example text"),
          },
        ],
      }),
    );
    // Use this ID in AddCommunicationToCase or CreateCase.
    console.log(response.attachmentSetId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, AddAttachmentsToSetconsulta AWS SDK for JavaScript API Reference.

`AddCommunicationToCase`

Il seguente esempio di codice mostra come utilizzare`AddCommunicationToCase`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { AddCommunicationToCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  let attachmentSetId;

  try {
    // Add a communication to a case.
    const response = await client.send(
      new AddCommunicationToCaseCommand({
        communicationBody: "Adding an attachment.",
        // Set value to an existing support case id.
        caseId: "CASE_ID",
        // Optional. Set value to an existing attachment set id to add attachments
 to the case.
        attachmentSetId,
      }),
    );
    console.log(response);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, AddCommunicationToCaseconsulta AWS SDK for JavaScript API Reference.

`CreateCase`

Il seguente esempio di codice mostra come utilizzare`CreateCase`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new case and log the case id.
    // Important: This creates a real support case in your account.
    const response = await client.send(
      new CreateCaseCommand({
        // The subject line of the case.
        subject: "IGNORE: Test case",
        // Use DescribeServices to find available service codes for each service.
        serviceCode: "service-quicksight-end-user",
        // Use DescribeSecurityLevels to find available severity codes for your
 support plan.
        severityCode: "low",
        // Use DescribeServices to find available category codes for each service.
        categoryCode: "end-user-support",
        // The main description of the support case.
        communicationBody: "This is a test. Please ignore.",
      }),
    );
    console.log(response.caseId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, CreateCaseconsulta AWS SDK for JavaScript API Reference.

`DescribeAttachment`

Il seguente esempio di codice mostra come utilizzare`DescribeAttachment`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeAttachmentCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the metadata and content of an attachment.
    const response = await client.send(
      new DescribeAttachmentCommand({
        // Set value to an existing attachment id.
        // Use DescribeCommunications or DescribeCases to find an attachment id.
        attachmentId: "ATTACHMENT_ID",
      }),
    );
    console.log(response.attachment?.fileName);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, DescribeAttachmentconsulta AWS SDK for JavaScript API Reference.

`DescribeCases`

Il seguente esempio di codice mostra come utilizzare`DescribeCases`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeCasesCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all of the unresolved cases in your account.
    // Filter or expand results by providing parameters to the DescribeCasesCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecasescommandinput.html
    const response = await client.send(new DescribeCasesCommand({}));
    const caseIds = response.cases.map((supportCase) => supportCase.caseId);
    console.log(caseIds);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, DescribeCasesconsulta AWS SDK for JavaScript API Reference.

`DescribeCommunications`

Il seguente esempio di codice mostra come utilizzare`DescribeCommunications`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeCommunicationsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all communications for the support case.
    // Filter results by providing parameters to the DescribeCommunicationsCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecommunicationscommandinput.html
    const response = await client.send(
      new DescribeCommunicationsCommand({
        // Set value to an existing case id.
        caseId: "CASE_ID",
      }),
    );
    const text = response.communications.map((item) => item.body).join("\n");
    console.log(text);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, DescribeCommunicationsconsulta AWS SDK for JavaScript API Reference.

`DescribeSeverityLevels`

Il seguente esempio di codice mostra come utilizzare`DescribeSeverityLevels`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DescribeSeverityLevelsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the list of severity levels.
    // The available values depend on the support plan for the account.
    const response = await client.send(new DescribeSeverityLevelsCommand({}));
    console.log(response.severityLevels);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, DescribeSeverityLevelsconsulta AWS SDK for JavaScript API Reference.

`ResolveCase`

Il seguente esempio di codice mostra come utilizzare`ResolveCase`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { ResolveCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

const main = async () => {
  try {
    const response = await client.send(
      new ResolveCaseCommand({
        caseId: "CASE_ID",
      }),
    );

    console.log(response.finalCaseStatus);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Per i dettagli sull'API, ResolveCaseconsulta AWS SDK for JavaScript API Reference.

Esempi di Systems Manager con SDK for JavaScript (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Systems Manager.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Nozioni di base

Salve Systems Manager

Il seguente esempio di codice mostra come iniziare a utilizzare Systems Manager.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { paginateListDocuments, SSMClient } from "@aws-sdk/client-ssm";

// Call ListDocuments and display the result.
export const main = async () => {
  const client = new SSMClient();
  const listDocumentsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListDocuments({ client }, { MaxResults: 5 });
    for await (const page of paginator) {
```

```
      listDocumentsPaginated.push(...page.DocumentIdentifiers);
    }
  } catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
  }

  for (const { Name, DocumentFormat, CreatedDate } of listDocumentsPaginated) {
    console.log(`${Name} - ${DocumentFormat} - ${CreatedDate}`);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Per i dettagli sull'API, consulta la ListDocumentssezione AWS SDK for JavaScript API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Creare una finestra di manutenzione.
- Modificare la pianificazione della finestra di manutenzione.
- Crea un documento.
- Invia un comando a un' EC2 istanza specificata.
- Crea un OpsItem.
- Aggiorna e risolvi il OpsItem.
- Eliminare la finestra di manutenzione OpsItem e il documento.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { fileURLToPath } from "node:url";
import {
  CreateDocumentCommand,
  CreateMaintenanceWindowCommand,
  CreateOpsItemCommand,
  DeleteDocumentCommand,
  DeleteMaintenanceWindowCommand,
  DeleteOpsItemCommand,
  DescribeOpsItemsCommand,
  DocumentAlreadyExists,
  OpsItemStatus,
  waitUntilCommandExecuted,
  CancelCommandCommand,
  paginateListCommandInvocations,
  SendCommandCommand,
  UpdateMaintenanceWindowCommand,
  UpdateOpsItemCommand,
  SSMClient,
} from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * @typedef {{
 *   ssmClient: import('@aws-sdk/client-ssm').SSMClient,
 *   documentName?: string
 *   maintenanceWindow?: string
 *   winId?: int
 *   ec2InstanceId?: string
```

```
 *   requestedDateTime?: Date
 *   opsItemId?: string
 *   askToDeleteResources?: boolean
 * }} State
 */

const defaultMaintenanceWindow = "ssm-maintenance-window";
const defaultDocumentName = "ssmdocument";
// The timeout duration is highly dependent on the specific setup and environment
 necessary. This example handles only the most common error cases, and uses a much
 shorter duration than most productions systems would use.
const COMMAND_TIMEOUT_DURATION_SECONDS = 30; // 30 seconds

const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `Welcome to the AWS Systems Manager SDK Getting Started scenario.
    This program demonstrates how to interact with Systems Manager using the AWS SDK
 for JavaScript V3.
    Systems Manager is the operations hub for your AWS applications and resources
 and a secure end-to-end management solution.
    The program's primary functions include creating a maintenance window, creating
 a document, sending a command to a document,
    listing documents, listing commands, creating an OpsItem, modifying an OpsItem,
 and deleting Systems Manager resources.
    Upon completion of the program, all AWS resources are cleaned up.
    Let's get started...`,
  { header: true },
);

const createMaintenanceWindow = new ScenarioOutput(
  "createMaintenanceWindow",
  "Step 1: Create a Systems Manager maintenance window.",
);

const getMaintenanceWindow = new ScenarioInput(
  "maintenanceWindow",
  "Please enter the maintenance window name:",
  { type: "input", default: defaultMaintenanceWindow },
);
```

```
export const sdkCreateMaintenanceWindow = new ScenarioAction(
  "sdkCreateMaintenanceWindow",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new CreateMaintenanceWindowCommand({
          Name: state.maintenanceWindow,
          Schedule: "cron(0 10 ? * MON-FRI *)", //The schedule of the maintenance
 window in the form of a cron or rate expression.
          Duration: 2, //The duration of the maintenance window in hours.
          Cutoff: 1, //The number of hours before the end of the maintenance window
 that Amazon Web Services Systems Manager stops scheduling new tasks for execution.
          AllowUnassociatedTargets: true, //Allow the maintenance window to run on
 managed nodes, even if you haven't registered those nodes as targets.
        }),
      );
      state.winId = response.WindowId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while creating the maintenance window. Please fix the
 error and try again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const modifyMaintenanceWindow = new ScenarioOutput(
  "modifyMaintenanceWindow",
  "Modify the maintenance window by changing the schedule.",
);

const sdkModifyMaintenanceWindow = new ScenarioAction(
  "sdkModifyMaintenanceWindow",
  async (/** @type {State} */ state) => {
    try {
      await state.ssmClient.send(
        new UpdateMaintenanceWindowCommand({
          WindowId: state.winId,
          Schedule: "cron(0 0 ? * MON *)",
        }),
      );
    } catch (caught) {
```

```
      console.error(caught.message);
      console.log(
        `An error occurred while modifying the maintenance window. Please fix the
 error and try again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const createSystemsManagerActions = new ScenarioOutput(
  "createSystemsManagerActions",
  "Create a document that defines the actions that Systems Manager performs on your
 EC2 instance.",
);

const getDocumentName = new ScenarioInput(
  "documentName",
  "Please enter the document: ",
  { type: "input", default: defaultDocumentName },
);

const sdkCreateSSMDoc = new ScenarioAction(
  "sdkCreateSSMDoc",
  async (/** @type {State} */ state) => {
    const contentData = `{
                "schemaVersion": "2.2",
                "description": "Run a simple shell command",
                "mainSteps": [
                    {
                        "action": "aws:runShellScript",
                        "name": "runEchoCommand",
                        "inputs": {
                          "runCommand": [
                            "echo 'Hello, world!'"
                          ]
                        }
                    }
                ]
            }`;
    try {
      await state.ssmClient.send(
        new CreateDocumentCommand({
          Content: contentData,
```

```
          Name: state.documentName,
          DocumentType: "Command",
        }),
      );
    } catch (caught) {
      console.log(`Exception type: (${typeof caught})`);
      if (caught instanceof DocumentAlreadyExists) {
        console.log("Document already exists. Continuing...\n");
      } else {
        console.error(caught.message);
        console.log(
          `An error occurred while creating the document. Please fix the error and
 try again. Error message: ${caught.message}`,
        );
        throw caught;
      }
    }
  },
);

const ec2HelloWorld = new ScenarioOutput(
  "ec2HelloWorld",
  `Now you have the option of running a command on an EC2 instance that echoes
 'Hello, world!'. In order to run this command, you must provide the instance ID
 of a Linux EC2 instance. If you do not already have a running Linux EC2 instance
 in your account, you can create one using the AWS console. For information about
 creating an EC2 instance, see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/
ec2-launch-instance-wizard.html.`,
);

const enterIdOrSkipEC2HelloWorld = new ScenarioInput(
  "enterIdOrSkipEC2HelloWorld",
  "Enter your EC2 InstanceId or press enter to skip this step: ",
  { type: "input", default: "" },
);

const sdkEC2HelloWorld = new ScenarioAction(
  "sdkEC2HelloWorld",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new SendCommandCommand({
          DocumentName: state.documentName,
          InstanceIds: [state.ec2InstanceId],
```

```
          TimeoutSeconds: COMMAND_TIMEOUT_DURATION_SECONDS,
        }),
      );
      state.CommandId = response.Command.CommandId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while sending the command. Please fix the error and try
 again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      state.enterIdOrSkipEC2HelloWorld === "",
  },
);

const sdkGetCommandTime = new ScenarioAction(
  "sdkGetCommandTime",
  async (/** @type {State} */ state) => {
    const listInvocationsPaginated = [];
    console.log(
      "Let's get the time when the specific command was sent to the specific managed
 node.",
    );

    console.log(
      `First, we'll wait for the command to finish executing. This may take up to
 ${COMMAND_TIMEOUT_DURATION_SECONDS} seconds.`,
    );
    const commandExecutedResult = waitUntilCommandExecuted(
      { client: state.ssmClient },
      {
        CommandId: state.CommandId,
        InstanceId: state.ec2InstanceId,
      },
    );
    // This is necessary because the TimeoutSeconds of SendCommandCommand is only
 for the delivery, not execution.
    try {
      await new Promise((_, reject) =>
        setTimeout(
```

```
          reject,
          COMMAND_TIMEOUT_DURATION_SECONDS * 1000,
          new Error("Command Timed Out"),
        ),
      );
    } catch (caught) {
      if (caught.message === "Command Timed Out") {
        commandExecutedResult.state = "TIMED_OUT";
      } else {
        throw caught;
      }
    }

    if (commandExecutedResult.state !== "SUCCESS") {
      console.log(
        `The command with id: ${state.CommandId} did not execute in the allotted
time. Canceling command.`,
      );
      state.ssmClient.send(
        new CancelCommandCommand({
          CommandId: state.CommandId,
        }),
      );
      state.enterIdOrSkipEC2HelloWorld === "";
      return;
    }

    for await (const page of paginateListCommandInvocations(
      { client: state.ssmClient },
      { CommandId: state.CommandId },
    )) {
      listInvocationsPaginated.push(...page.CommandInvocations);
    }
    /**
     * @type {import('@aws-sdk/client-ssm').CommandInvocation}
     */
    const commandInvocation = listInvocationsPaginated.shift(); // Because the call
was made with CommandId, there's only one result, so shift it off.
    state.requestedDateTime = commandInvocation.RequestedDateTime;

    console.log(
      `The command invocation happened at: ${state.requestedDateTime}.`,
    );
  },
```

```
  {
    skipWhen: (/** @type {State} */ state) =>
      state.enterIdOrSkipEC2HelloWorld === "",
  },
);

const createSSMOpsItem = new ScenarioOutput(
  "createSSMOpsItem",
  `Now we will create a Systems Manager OpsItem. An OpsItem is a feature provided by
 the Systems Manager service. It is a type of operational data item that allows you
 to manage and track various operational issues, events, or tasks within your AWS
 environment.
You can create OpsItems to track and manage operational issues as they arise. For
 example, you could create an OpsItem whenever your application detects a critical
 error or an anomaly in your infrastructure.`,
);

const sdkCreateSSMOpsItem = new ScenarioAction(
  "sdkCreateSSMOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new CreateOpsItemCommand({
          Description: "Created by the System Manager Javascript API",
          Title: "Disk Space Alert",
          Source: "EC2",
          Category: "Performance",
          Severity: "2",
        }),
      );
      state.opsItemId = response.OpsItemId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while creating the ops item. Please fix the error and try
 again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const updateOpsItem = new ScenarioOutput(
  "updateOpsItem",
```

```
  (/** @type {State} */ state) =>
    `Now we will update the OpsItem: ${state.opsItemId}`,
);

const sdkUpdateOpsItem = new ScenarioAction(
  "sdkUpdateOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const _response = await state.ssmClient.send(
        new UpdateOpsItemCommand({
          OpsItemId: state.opsItemId,
          Description: `An update to ${state.opsItemId}`,
        }),
      );
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while updating the ops item. Please fix the error and try
 again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const getOpsItemStatus = new ScenarioOutput(
  "getOpsItemStatus",
  (/** @type {State} */ state) =>
    `Now we will get the status of the OpsItem: ${state.opsItemId}`,
);

const sdkOpsItemStatus = new ScenarioAction(
  "sdkGetOpsItemStatus",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new DescribeOpsItemsCommand({
          OpsItemId: state.opsItemId,
        }),
      );
      state.opsItemStatus = response.OpsItemStatus;
    } catch (caught) {
      console.error(caught.message);
      console.log(
```

```
        `An error occurred while describing the ops item. Please fix the error and
 try again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const resolveOpsItem = new ScenarioOutput(
  "resolveOpsItem",
  (/** @type {State} */ state) =>
    `Now we will resolve the OpsItem: ${state.opsItemId}`,
);

const sdkResolveOpsItem = new ScenarioAction(
  "sdkResolveOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const _response = await state.ssmClient.send(
        new UpdateOpsItemCommand({
          OpsItemId: state.opsItemId,
          Status: OpsItemStatus.RESOLVED,
        }),
      );
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while updating the ops item. Please fix the error and try
 again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const askToDeleteResources = new ScenarioInput(
  "askToDeleteResources",
  "Would you like to delete the Systems Manager resources created during this
 example run?",
  { type: "confirm" },
);

const confirmDeleteChoice = new ScenarioOutput(
  "confirmDeleteChoice",
```

```
  (/** @type {State} */ state) => {
    if (state.askToDeleteResources) {
      return "You chose to delete the resources.";
    }
    return "The Systems Manager resources will not be deleted. Please delete them
 manually to avoid charges.";
  },
);

export const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    try {
      await state.ssmClient.send(
        new DeleteOpsItemCommand({
          OpsItemId: state.opsItemId,
        }),
      );
      console.log(`The ops item: ${state.opsItemId} was successfully deleted.`);
    } catch (caught) {
      console.log(
        `There was a problem deleting the ops item: ${state.opsItemId}. Please
 delete it manually. Error: ${caught.message}`,
      );
    }

    try {
      await state.ssmClient.send(
        new DeleteMaintenanceWindowCommand({
          Name: state.maintenanceWindow,
          WindowId: state.winId,
        }),
      );
      console.log(
        `The maintenance window: ${state.maintenanceWindow} was successfully
 deleted.`,
      );
    } catch (caught) {
      console.log(
        `There was a problem deleting the maintenance window: ${state.opsItemId}.
 Please delete it manually. Error: ${caught.message}`,
      );
    }
```

```
    try {
      await state.ssmClient.send(
        new DeleteDocumentCommand({
          Name: state.documentName,
        }),
      );
      console.log(
        `The document: ${state.documentName} was successfully deleted.`,
      );
    } catch (caught) {
      console.log(
        `There was a problem deleting the document: ${state.documentName}. Please
 delete it manually. Error: ${caught.message}`,
      );
    }
  },
  { skipWhen: (/** @type {{}} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the Systems Manager Basics scenario for the AWS Javascript SDK v3.
 Thank you!",
);

const myScenario = new Scenario(
  "SSM Basics",
  [
    greet,
    pressEnter,
    createMaintenanceWindow,
    getMaintenanceWindow,
    sdkCreateMaintenanceWindow,
    modifyMaintenanceWindow,
    pressEnter,
    sdkModifyMaintenanceWindow,
    createSystemsManagerActions,
    getDocumentName,
    sdkCreateSSMDoc,
    ec2HelloWorld,
    enterIdOrSkipEC2HelloWorld,
    sdkEC2HelloWorld,
    sdkGetCommandTime,
    pressEnter,
```

```
    createSSMOpsItem,
    pressEnter,
    sdkCreateSSMOpsItem,
    updateOpsItem,
    pressEnter,
    sdkUpdateOpsItem,
    getOpsItemStatus,
    pressEnter,
    sdkOpsItemStatus,
    resolveOpsItem,
    pressEnter,
    sdkResolveOpsItem,
    askToDeleteResources,
    confirmDeleteChoice,
    sdkDeleteResources,
    goodbye,
  ],
  { ssmClient: new SSMClient({}) },
);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for JavaScript .
  - CreateDocument

- CreateMaintenanceWindow
- CreateOpsItem
- DeleteMaintenanceWindow
- ListCommandInvocations
- SendCommand
- UpdateOpsItem

Azioni

`CreateDocument`

Il seguente esempio di codice mostra come usare`CreateDocument`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateDocumentCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ content, name, documentType }) => {
  const client = new SSMClient({});
  try {
    const { documentDescription } = await client.send(
      new CreateDocumentCommand({
        Content: content, // The content for the new SSM document. The content must
 not exceed 64KB.
        Name: name,
        DocumentType: documentType, // Document format type can be JSON, YAML, or
 TEXT. The default format is JSON.
      }),
```

```
    );
    console.log("Document created successfully.");
    return { DocumentDescription: documentDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DocumentAlreadyExists") {
      console.warn(`${caught.message}. Did you provide a new document name?`);
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, consulta la CreateDocumentsezione AWS SDK for JavaScript API Reference.

`CreateMaintenanceWindow`

Il seguente esempio di codice mostra come utilizzare`CreateMaintenanceWindow`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM maintenance window.
 * @param {{ name: string, allowUnassociatedTargets: boolean, duration: number,
 cutoff: number, schedule: string, description?: string }}
 */
export const main = async ({
  name,
  allowUnassociatedTargets, // Allow the maintenance window to run on managed nodes,
 even if you haven't registered those nodes as targets.
  duration, // The duration of the maintenance window in hours.
```

```
  cutoff, // The number of hours before the end of the maintenance window that
 Amazon Web Services Systems Manager stops scheduling new tasks for execution.
  schedule, // The schedule of the maintenance window in the form of a cron or rate
 expression.
  description = undefined,
}) => {
  const client = new SSMClient({});

  try {
    const { windowId } = await client.send(
      new CreateMaintenanceWindowCommand({
        Name: name,
        Description: description,
        AllowUnassociatedTargets: allowUnassociatedTargets, // Allow the maintenance
 window to run on managed nodes, even if you haven't registered those nodes as
 targets.
        Duration: duration, // The duration of the maintenance window in hours.
        Cutoff: cutoff, // The number of hours before the end of the maintenance
 window that Amazon Web Services Systems Manager stops scheduling new tasks for
 execution.
        Schedule: schedule, // The schedule of the maintenance window in the form of
 a cron or rate expression.
      }),
    );
    console.log(`Maintenance window created with Id: ${windowId}`);
    return { WindowId: windowId };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, consulta la CreateMaintenanceWindowsezione AWS SDK for JavaScript API Reference.

`CreateOpsItem`

Il seguente esempio di codice mostra come utilizzare`CreateOpsItem`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { CreateOpsItemCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM OpsItem.
 * @param {{ title: string, source: string, category?: string, severity?: string }}
 */
export const main = async ({
  title,
  source,
  category = undefined,
  severity = undefined,
}) => {
  const client = new SSMClient({});
  try {
    const { opsItemArn, opsItemId } = await client.send(
      new CreateOpsItemCommand({
        Title: title,
        Source: source, // The origin of the OpsItem, such as Amazon EC2 or Systems
 Manager.
        Category: category,
        Severity: severity,
      }),
    );
    console.log(`Ops item created with id: ${opsItemId}`);
    return { OpsItemArn: opsItemArn, OpsItemId: opsItemId };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, consulta la CreateOpsItemsezione AWS SDK for JavaScript API Reference.

`DeleteDocument`

Il seguente esempio di codice mostra come utilizzare`DeleteDocument`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteDocumentCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Delete an SSM document.
 * @param {{ documentName: string }}
 */
export const main = async ({ documentName }) => {
  const client = new SSMClient({});
  try {
    await client.send(new DeleteDocumentCommand({ Name: documentName }));
    console.log(`Document '${documentName}' deleted.`);
    return { Deleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, consulta la DeleteDocumentsezione AWS SDK for JavaScript API Reference.

`DeleteMaintenanceWindow`

Il seguente esempio di codice mostra come utilizzare`DeleteMaintenanceWindow`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { DeleteMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Delete an SSM maintenance window.
 * @param {{ windowId: string }}
 */
export const main = async ({ windowId }) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new DeleteMaintenanceWindowCommand({ WindowId: windowId }),
    );
    console.log(`Maintenance window '${windowId}' deleted.`);
    return { Deleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, consulta la DeleteMaintenanceWindowsezione AWS SDK for JavaScript API Reference.

`DescribeOpsItems`

Il seguente esempio di codice mostra come utilizzare`DescribeOpsItems`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import {
  OpsItemFilterOperator,
  OpsItemFilterKey,
  paginateDescribeOpsItems,
  SSMClient,
} from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Describe SSM OpsItems.
 * @param {{ opsItemId: string }}
 */
export const main = async ({ opsItemId }) => {
  const client = new SSMClient({});
  try {
    const describeOpsItemsPaginated = [];
    for await (const page of paginateDescribeOpsItems(
      { client },
      {
        OpsItemFilters: {
          Key: OpsItemFilterKey.OPSITEM_ID,
          Operator: OpsItemFilterOperator.EQUAL,
          Values: opsItemId,
        },
      },
    )) {
      describeOpsItemsPaginated.push(...page.OpsItemSummaries);
```

```
    }
    console.log("Here are the ops items:");
    console.log(describeOpsItemsPaginated);
    return { OpsItemSummaries: describeOpsItemsPaginated };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    }
    throw caught;
  }
};
```

- Per i dettagli sull'API, consulta la DescribeOpsItemssezione AWS SDK for JavaScript API Reference.

`ListCommandInvocations`

Il seguente esempio di codice mostra come utilizzare`ListCommandInvocations`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { paginateListCommandInvocations, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * List SSM command invocations on an instance.
 * @param {{ instanceId: string }}
 */
export const main = async ({ instanceId }) => {
  const client = new SSMClient({});
  try {
    const listCommandInvocationsPaginated = [];
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListCommandInvocations(
```

```
      { client },
      {
        InstanceId: instanceId,
      },
    );
    for await (const page of paginator) {
      listCommandInvocationsPaginated.push(...page.CommandInvocations);
    }
    console.log("Here is the list of command invocations:");
    console.log(listCommandInvocationsPaginated);
    return { CommandInvocations: listCommandInvocationsPaginated };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
      console.warn(`${caught.message}. Did you provide a valid instance ID?`);
    }
    throw caught;
  }
};
```

- Per i dettagli sull'API, consulta la ListCommandInvocationssezione AWS SDK for JavaScript API Reference.

`SendCommand`

Il seguente esempio di codice mostra come utilizzare`SendCommand`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { SendCommandCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Send an SSM command to a managed node.
 * @param {{ documentName: string }}
```

```
 */
export const main = async ({ documentName }) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new SendCommandCommand({
        DocumentName: documentName,
      }),
    );
    console.log("Command sent successfully.");
    return { Success: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
      console.warn(`${caught.message}. Did you provide a valid document name?`);
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, consulta la SendCommandsezione AWS SDK for JavaScript API Reference.

`UpdateMaintenanceWindow`

Il seguente esempio di codice mostra come utilizzare`UpdateMaintenanceWindow`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { UpdateMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Update an SSM maintenance window.
```

```
 * @param {{ windowId: string, allowUnassociatedTargets?: boolean, duration?:
 number, enabled?: boolean, name?: string, schedule?: string }}
 */
export const main = async ({
  windowId,
  allowUnassociatedTargets = undefined, //Allow the maintenance window to run on
 managed nodes, even if you haven't registered those nodes as targets.
  duration = undefined, //The duration of the maintenance window in hours.
  enabled = undefined,
  name = undefined,
  schedule = undefined, //The schedule of the maintenance window in the form of a
 cron or rate expression.
}) => {
  const client = new SSMClient({});
  try {
    const { opsItemArn, opsItemId } = await client.send(
      new UpdateMaintenanceWindowCommand({
        WindowId: windowId,
        AllowUnassociatedTargets: allowUnassociatedTargets,
        Duration: duration,
        Enabled: enabled,
        Name: name,
        Schedule: schedule,
      }),
    );
    console.log("Maintenance window updated.");
    return { OpsItemArn: opsItemArn, OpsItemId: opsItemId };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
      console.warn(`${caught.message}. Are these values correct?`);
    } else {
      throw caught;
    }
  }
};
```

- Per i dettagli sull'API, consulta la UpdateMaintenanceWindowsezione AWS SDK for JavaScript API Reference.

`UpdateOpsItem`

Il seguente esempio di codice mostra come utilizzare`UpdateOpsItem`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { UpdateOpsItemCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Update an SSM OpsItem.
 * @param {{ opsItemId: string, status?: OpsItemStatus }}
 */
export const main = async ({
  opsItemId,
  status = undefined, // The OpsItem status. Status can be Open, In Progress, or
 Resolved
}) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new UpdateOpsItemCommand({
        OpsItemId: opsItemId,
        Status: status,
      }),
    );
    console.log("Ops item updated.");
    return { Success: true };
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "OpsItemLimitExceededException"
    ) {
      console.warn(
        `Couldn't create ops item because you have exceeded your open OpsItem limit.
 ${caught.message}.`,
      );
    } else {
      throw caught;
    }
```

```
  }
};
```

- Per i dettagli sull'API, consulta la UpdateOpsItemsezione AWS SDK for JavaScript API Reference.

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.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

Scenari

Creazione di un'applicazione Amazon Textract explorer

L'esempio di codice seguente mostra come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

SDK per (v3 JavaScript)

> 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 SQS) sottoscritta a un argomento 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
- Simple Storage Service (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.

SDK per JavaScript (v3)

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in. GitHub I seguenti estratti mostrano come AWS SDK for JavaScript viene utilizzato all'interno delle funzioni Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
```

```
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
```

```
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
```

```
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);
```

```
  return { translated_text: TranslatedText };
};
```

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Esempi di Amazon Transcribe con SDK JavaScript for (v3)

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for JavaScript (v3) con Amazon Transcribe.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni
- Scenari

Azioni

`DeleteMedicalTranscriptionJob`

Il seguente esempio di codice mostra come usare. `DeleteMedicalTranscriptionJob`

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
import { 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 i dettagli sull'API, consulta la DeleteMedicalTranscriptionJobsezione AWS SDK for JavaScript API Reference.

`DeleteTranscriptionJob`

Il seguente esempio di codice mostra come utilizzare`DeleteTranscriptionJob`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eliminare un 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 i dettagli sull'API, consulta la DeleteTranscriptionJobsezione AWS SDK for JavaScript API Reference.

`ListMedicalTranscriptionJobs`

Il seguente esempio di codice mostra come utilizzare`ListMedicalTranscriptionJobs`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
import { 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 i dettagli sull'API, consulta la ListMedicalTranscriptionJobssezione AWS SDK for JavaScript API Reference.

`ListTranscriptionJobs`

Il seguente esempio di codice mostra come utilizzare`ListTranscriptionJobs`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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 i dettagli sull'API, consulta la ListTranscriptionJobssezione AWS SDK for JavaScript API Reference.

`StartMedicalTranscriptionJob`

Il seguente esempio di codice mostra come utilizzare`StartMedicalTranscriptionJob`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea il client.

```
import { 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 i dettagli sull'API, consulta la StartMedicalTranscriptionJobsezione AWS SDK for JavaScript API Reference.

`StartTranscriptionJob`

Il seguente esempio di codice mostra come utilizzare`StartTranscriptionJob`.

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

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 i dettagli sull'API, consulta la StartTranscriptionJobsezione AWS SDK for JavaScript API Reference.

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.

SDK per JavaScript (v3)

Mostra come utilizzare Amazon Transcribe per creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia i risultati per e-mail tramite Amazon Simple Email Service (Amazon SES).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

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.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

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.

SDK per JavaScript (v3)

Mostra come utilizzare Amazon Transcribe per creare un'applicazione che registra, trascrive e traduce l'audio in tempo reale e invia i risultati per e-mail tramite Amazon Simple Email Service (Amazon SES).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Creazione di un chatbot Amazon Lex

L'esempio di codice seguente mostra come creare un chatbot per coinvolgere i visitatori del sito web.

SDK per JavaScript (v3)

Mostra come utilizzare l'API Amazon Lex per creare un chatbot all'interno di un'applicazione web per coinvolgere i visitatori del sito web.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo Costruire un chatbot Amazon Lex nella guida per gli AWS SDK for JavaScript sviluppatori.

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

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.

SDK per JavaScript (v3)

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in GitHub. I seguenti estratti mostrano come AWS SDK for JavaScript viene utilizzato all'interno delle funzioni Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
```

```
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
```

```
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
```

```
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });
```

```
  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Sicurezza per questo AWS prodotto o servizio

La sicurezza cloud di Amazon Web Services (AWS) è la priorità più alta. In qualità di 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 Servizi AWS utilizzati. Per maggiori informazioni sulla privacy dei dati, consulta le Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al AWS Modello di responsabilità condivisa e GDPR nel AWS Blog sulla sicurezza.

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- SSL/TLS Da utilizzare per comunicare con AWS le risorse. È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con AWS CloudTrail. Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta Lavorare con i CloudTrail percorsi nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-3.

Ti consigliamo 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 lavora con questo AWS prodotto o servizio o altro Servizi AWS utilizzando la console, l'API o. AWS CLI 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. Quando si fornisce un URL a un server esterno, suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la richiesta al server.

Identity and Access Management

AWS Identity and Access Management (IAM) è un software Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori

IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle autorizzazioni) a utilizzare le risorse. AWS IAM è uno Servizio AWS strumento che puoi utilizzare senza costi aggiuntivi.

Argomenti

Destinatari

Il modo in cui usi AWS Identity and Access Management (IAM) varia a seconda del lavoro che AWS svolgi.

Utente del servizio: se lo utilizzi Servizi AWS per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più AWS funzionalità per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell’accesso consente di richiedere le autorizzazioni corrette all’amministratore. Se non riesci ad accedere a una funzionalità di AWS, consulta Risoluzione dei problemi di AWS identità e accesso o consulta la guida per l'utente della funzionalità Servizio AWS che stai utilizzando.

Amministratore del servizio: se sei responsabile delle AWS risorse della tua azienda, probabilmente hai pieno accesso a AWS. È tuo compito determinare a quali AWS funzionalità e risorse devono accedere gli utenti del servizio. Devi quindi inviare le richieste all’amministratore IAM per modificare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per saperne di più su come la tua azienda può utilizzare IAM con AWS, consulta la guida per l'utente del Servizio AWS software che stai utilizzando.

Amministratore IAM: un amministratore IAM potrebbe essere interessato a ottenere dei dettagli su come scrivere policy per gestire l'accesso a AWS. Per visualizzare esempi di policy AWS basate sull'identità che puoi utilizzare in IAM, consulta la guida per l'utente di quella Servizio AWS che stai utilizzando.

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi autenticarti come utente IAM o assumendo un ruolo IAM. Utente root dell'account AWS

Puoi accedere come identità federata utilizzando credenziali provenienti da una fonte di identità come AWS IAM Identity Center (IAM Identity Center), autenticazione Single Sign-On o credenziali. Google/ Facebook Per ulteriori informazioni sull'accesso, consulta Come accedere all' Account AWS nella Guida per l'utente di Accedi ad AWS .

Per l'accesso programmatico, AWS fornisce un SDK e una CLI per firmare crittograficamente le richieste. Per ulteriori informazioni, consulta AWS Signature Version 4 per le richieste API nella Guida per l'utente di IAM.

Account AWS utente root

Quando si crea un Account AWS, si inizia con un'identità di accesso denominata utente Account AWS root che ha accesso completo a tutte Servizi AWS le risorse. Consigliamo vivamente di non utilizzare l'utente root per le attività quotidiane. Per le attività che richiedono le credenziali dell'utente root, consulta Attività che richiedono le credenziali dell'utente root nella Guida per l'utente IAM.

Identità federata

Come procedura ottimale, richiedi agli utenti umani di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente della directory aziendale, del provider di identità Web o Directory Service che accede Servizi AWS utilizzando le credenziali di una fonte di identità. Le identità federate assumono ruoli che forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, si consiglia di utilizzare AWS IAM Identity Center. Per ulteriori informazioni, consulta Che cos'è il Centro identità IAM? nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un utente IAM è una identità che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ti consigliamo di utilizzare credenziali temporanee invece di utenti IAM con credenziali a lungo termine. Per ulteriori informazioni, consulta Richiedere agli utenti umani di utilizzare la

federazione con un provider di identità per accedere AWS utilizzando credenziali temporanee nella Guida per l'utente IAM.

Un gruppo IAM specifica una raccolta di utenti IAM e semplifica la gestione delle autorizzazioni per gestire gruppi di utenti di grandi dimensioni. Per ulteriori informazioni, consulta Casi d'uso per utenti IAM nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità con autorizzazioni specifiche che fornisce credenziali temporanee. Puoi assumere un ruolo passando da un ruolo utente a un ruolo IAM (console) o chiamando un'operazione AWS CLI o AWS API. Per ulteriori informazioni, consulta Metodi per assumere un ruolo nella Guida per l'utente di IAM.

I ruoli IAM sono utili per l'accesso federato degli utenti, le autorizzazioni utente IAM temporanee, l'accesso tra account, l'accesso tra servizi e le applicazioni in esecuzione su Amazon. EC2 Per maggiori informazioni, consultare Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Gestione dell'accesso tramite policy

Puoi controllare l'accesso AWS creando policy e collegandole a identità o risorse. AWS Una policy definisce le autorizzazioni quando è associata a un'identità o a una risorsa. AWS valuta queste politiche quando un preside effettua una richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per maggiori informazioni sui documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Utilizzando le policy, gli amministratori specificano chi ha accesso a cosa definendo quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Un amministratore IAM crea le policy IAM e le aggiunge ai ruoli, che gli utenti possono quindi assumere. Le policy IAM definiscono le autorizzazioni indipendentemente dal metodo utilizzato per eseguirle.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile collegare a un'identità (utente, gruppo o ruolo). Tali policy controllano le operazioni autorizzate per l'identità, nonché le risorse e le condizioni in cui possono essere eseguite. Per informazioni su come

creare una policy basata su identità, consultare Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Le policy basate su identità possono essere policy in linea (con embedding direttamente in una singola identità) o policy gestite (policy autonome collegate a più identità). Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scegliere tra policy gestite e policy in linea nella Guida per l'utente di IAM.

Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi includono le policy di trust dei ruoli IAM 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. In una policy basata sulle risorse è obbligatorio specificare un'entità principale.

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Liste 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. ACLs sono 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 informazioni ACLs, consulta la panoramica della lista di controllo degli accessi (ACL) nella Amazon Simple Storage Service Developer Guide.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi che possono impostare le autorizzazioni massime concesse dai tipi di policy più comuni:

- Limiti delle autorizzazioni: imposta il numero massimo di autorizzazioni che una policy basata su identità ha la possibilità di concedere a un'entità IAM. Per ulteriori informazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente di IAM.
- Politiche di controllo del servizio (SCPs): specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa in. AWS Organizations Per ulteriori informazioni, consultare Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations .

- Politiche di controllo delle risorse (RCPs): imposta le autorizzazioni massime disponibili per le risorse nei tuoi account. Per ulteriori informazioni, consulta Politiche di controllo delle risorse (RCPs) nella Guida per l'AWS Organizations utente.
- Policy di sessione: policy avanzate passate come parametro quando si crea una sessione temporanea per un ruolo o un utente federato. Per maggiori informazioni, consultare Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando a una richiesta si applicano più tipi di policy, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella IAM User Guide.

Come Servizi AWS lavorare con IAM

Per avere una visione di alto livello di come Servizi AWS funziona la maggior parte delle funzionalità IAM, consulta AWS i servizi che funzionano con IAM nella IAM User Guide.

Per scoprire come utilizzare uno specifico Servizio AWS con IAM, consulta la sezione sulla sicurezza della Guida per l'utente del servizio pertinente.

Risoluzione dei problemi di AWS identità e accesso

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con un AWS IAM.

Argomenti

- Non sono autorizzato a eseguire alcuna azione in AWS
- Non sono autorizzato a eseguire iam: PassRole
- Voglio consentire a persone esterne a me di accedere Account AWS alle mie AWS risorse

Non sono autorizzato a eseguire alcuna azione in AWS

Se ricevi un errore che indica che non sei autorizzato a eseguire un'operazione, le tue policy devono essere aggiornate per poter eseguire l'operazione.

L'errore di esempio seguente si verifica quando l'utente IAM `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa *`my-example-widget`* fittizia ma non dispone di autorizzazioni `awes:`*`GetWidget`* fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In questo caso, la policy per l'utente `mateojackson` deve essere aggiornata per consentire l'accesso alla risorsa *`my-example-widget`* utilizzando l'azione `awes:`*`GetWidget`*.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un errore che indica che non sei autorizzato a eseguire l'operazione `iam:PassRole`, le tue policy devono essere aggiornate per poter passare un ruolo a AWS.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

L'errore di esempio seguente si verifica quando un utente IAM denominato `marymajor` cerca di utilizzare la console per eseguire un'operazione in AWS. Tuttavia, l'operazione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per trasmettere il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie AWS risorse

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo.

Per 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 maggiori informazioni, consulta gli argomenti seguenti:

- Per sapere se AWS supporta queste funzionalità, consulta. Come Servizi AWS lavorare con IAM
- Per scoprire come fornire l'accesso alle tue risorse attraverso Account AWS le risorse di tua proprietà, consulta Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà nella IAM User Guide.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta Fornire l'accesso a soggetti Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso a utenti autenticati esternamente (federazione delle identità) nella Guida per l'utente IAM.
- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente di IAM.

Convalida della conformità per questo AWS prodotto o servizio

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione Scope by Compliance Program Servizi AWS e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di AWS conformità Programmi di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in AWS Artifact .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. Per ulteriori informazioni sulla responsabilità di conformità durante l'utilizzo Servizi AWS, consulta la Documentazione AWS sulla sicurezza.

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 è possibile progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, vedere Global Infrastructure.AWS

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Sicurezza dell'infrastruttura per questo AWS prodotto o servizio

Questo AWS prodotto o servizio utilizza servizi gestiti ed è pertanto protetto dalla sicurezza di rete AWS globale. Per informazioni sui servizi AWS di sicurezza e su come AWS protegge l'infrastruttura, consulta AWS Cloud Security. Per progettare il tuo AWS ambiente utilizzando le migliori pratiche per la sicurezza dell'infrastruttura, vedi Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Utilizzate chiamate API AWS pubblicate per accedere a questo AWS Prodotto o Servizio attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. In alternativa è possibile utilizzare AWS Security Token Service (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

Questo AWS prodotto o servizio segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Applica una versione TLS minima

Per aumentare la sicurezza durante la comunicazione con AWS i servizi, configurali AWS SDK for JavaScript per utilizzare TLS 1.2 o versioni successive.

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.

> **Important**
>
> A partire dal 10 giugno 2024, abbiamo annunciato che TLS 1.3 è disponibile sugli endpoint delle API di AWS servizio in ciascuna delle regioni. AWS La versione AWS SDK for JavaScript v3 non negozia la versione TLS stessa. Utilizza invece la versione TLS determinata da Node.js, che è configurabile tramite. `https.Agent` AWS consiglia di utilizzare l'attuale versione Active LTS di Node.js.

Verificare e applicare TLS in Node.js

Quando si utilizza il AWS SDK for JavaScript con Node.js, il livello di sicurezza Node.js sottostante viene utilizzato per impostare la versione TLS.

Node.js 12.0.0 e versioni successive utilizzano una versione minima di OpenSSL 1.1.1b, che supporta TLS 1.3. Per impostazione predefinita, Node.js utilizza TLS 1.3 quando disponibile. È possibile specificare esplicitamente una versione diversa, se necessario.

Verificare la versione di OpenSSL e TLS

Per ottenere la versione di OpenSSL utilizzata da Node.js sul computer, eseguire il seguente comando.

```
node -p process.versions
```

La versione di OpenSSL nell'elenco è la versione utilizzata da Node.js, come mostrato nell'esempio seguente.

```
openssl: '1.1.1b'
```

Per ottenere la versione di TLS utilizzata da Node.js sul computer, avviare la shell Node ed eseguire i seguenti comandi, in ordine.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

L'ultimo comando restituisce la versione TLS, come mostrato nell'esempio seguente.

```
'TLSv1.3'
```

Node.js utilizza per impostazione predefinita questa versione di TLS e tenta di negoziare un'altra versione di TLS se una chiamata non ha esito positivo.

Verifica delle versioni TLS minime e massime supportate

Gli sviluppatori possono verificare le versioni TLS minime e massime supportate in Node.js utilizzando lo script seguente:

```
import tls from "tls";
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +
 tls.DEFAULT_MAX_VERSION);
```

L'ultimo comando restituisce la versione TLS minima e massima predefinita, come illustrato nell'esempio seguente.

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

Applicazione di una versione minima di TLS

Node.js negozia una versione di TLS quando una chiamata non riesce. È possibile applicare la versione TLS minima consentita durante questa negoziazione, sia quando si esegue uno script dalla riga di comando che per richiesta nel codice. JavaScript

Per specificare la versione TLS minima dalla riga di comando, è necessario utilizzare Node.js versione 11.4.0 o successiva. Per installare una versione specifica di Node.js, installa prima Node Version Manager (nvm) utilizzando i passaggi disponibili in Installazione e aggiornamento del gestore delle versioni di Node. Quindi eseguire i seguenti comandi per installare e utilizzare una versione specifica di Node.js.

```
nvm install 11
nvm use 11
```

Enforce TLS 1.2

Per stabilire che TLS 1.2 sia la versione minima consentita, specificare l'argomento `--tls-min-v1.2` durante l'esecuzione dello script, come mostrato nell'esempio seguente.

```
node --tls-min-v1.2 yourScript.js
```

Per specificare la versione TLS minima consentita per una richiesta specifica nel JavaScript codice, utilizzate il `minVersion` 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(
            {
                minVersion: 'TLSv1.2'
            }
        )
    })
});
```

Enforce TLS 1.3

Per far sì che TLS 1.3 sia la versione minima consentita, specificate l'`--tls-min-v1.3`argomento durante l'esecuzione dello script, come illustrato nell'esempio seguente.

```
node --tls-min-v1.3 yourScript.js
```

Per specificare la versione TLS minima consentita per una richiesta specifica nel JavaScript codice, utilizzate il `minVersion` 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(
            {
                minVersion: 'TLSv1.3'
            }
        )
    })
});
```

Verificare e applicare TLS in uno script del browser

Quando utilizzate l'SDK per JavaScript in uno script del browser, le impostazioni del browser controllano la versione di TLS utilizzata. La versione di TLS utilizzata dal browser non può essere individuata o impostata dallo script e deve essere configurata dall'utente. Per verificare e applicare la versione di TLS utilizzata in uno script del browser, consultare le istruzioni relative al browser specifico.

Microsoft Internet Explorer

1. Apri Internet Explorer.
2. Dalla barra dei menu, scegli Strumenti - Opzioni Internet - scheda Avanzate.
3. Scorri verso il basso fino alla categoria Sicurezza, seleziona manualmente la casella di opzione Usa TLS 1.2.
4. Fai clic su OK.
5. Chiudi il browser e riavvia Internet Explorer.

Microsoft Edge

1. Nella casella di ricerca del menu 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 User TLS 1.2.
5. Fai clic su OK.

Google Chrome

1. Apri Google Chrome.
2. Fai clic su Alt F e seleziona Impostazioni.
3. Scorri verso il basso e seleziona Mostra impostazioni avanzate... .
4. Scorri verso il basso fino alla sezione Sistema e fai clic su Apri impostazioni proxy... .
5. Seleziona la scheda Avanzate.
6. Scorri verso il basso fino alla categoria Sicurezza, seleziona manualmente la casella di opzione Usa TLS 1.2.
7. Fai clic su OK.
8. Chiudi il browser e riavvia Google Chrome.

Mozilla Firefox

1. Apri Firefox.
2. Nella barra degli indirizzi, digita about:config e premi Invio.
3. Nel campo Cerca, inserisci tls. Trova e fai doppio clic sulla voce security.tls.version.min.
4. Imposta il valore intero su 3 per forzare il protocollo TLS 1.2 a diventare quello predefinito.
5. Fai clic su OK.
6. Chiudi il browser e riavvia Mozilla Firefox.

Apple Safari

Non ci sono opzioni per abilitare i protocolli SSL. Se utilizzi la versione 7 o successiva di Safari, TLS 1.2 viene abilitato automaticamente.

Recupero della versione TLS nelle richieste v3 AWS SDK for JavaScript

Puoi registrare la versione TLS utilizzata in una richiesta AWS SDK con il seguente script:

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
import tls from "tls";

const client = new S3Client({ region: "us-east-1" });

const tlsSocket = new tls.TLSSocket();

client.middlewareStack.add((next, context) => async (args) => {
  console.log(`Using TLS version: ${tlsSocket.getProtocol()}`);
    return next(args);
});
```

L'ultimo comando restituisce la versione TLS in uso, come illustrato nell'esempio seguente.

```
Using TLS version: TLSv1.3
```

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 dell'SDK for. JavaScript

Migra il codice su SDK per v3 usando codemod JavaScript

AWS SDK for JavaScript la versione 3 (v3) include interfacce modernizzate per le configurazioni e le utilità dei client, che includono credenziali, caricamento multiparte di Amazon S3, client di documenti DynamoDB, camerieri e altro ancora. Puoi trovare cosa è cambiato nella v2 e nelle versioni equivalenti della v3 per ogni modifica nella guida alla migrazione sul repository. AWS SDK for JavaScript GitHub

Per sfruttare appieno la AWS SDK for JavaScript v3, consigliamo di utilizzare gli script codemod descritti di seguito.

Usa codemod per migrare il codice v2 esistente

La raccolta di script codemod in aws-sdk-js-codemodaiuta a migrare l'applicazione esistente AWS SDK for JavaScript (v2) per utilizzare 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 dell'SDK for JavaScript (v3) contiene le seguenti nuove funzionalità.

Pacchetti modularizzati

Gli utenti possono ora utilizzare un pacchetto separato per ogni servizio.

Nuovo stack di middleware

Gli utenti possono ora utilizzare uno stack middleware per controllare il ciclo di vita di una chiamata operativa.

Inoltre, l'SDK è integrato, il che presenta molti vantaggi TypeScript, come la digitazione statica.

> **Important**
>
> Gli esempi di codice per v3 in questa guida sono scritti in ECMAScript 6 (). ES6 ES6offre una nuova sintassi e nuove funzionalità per rendere il codice più moderno e leggibile e fare di più.

ES6 richiede 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 ES6Sintassi /CommonJS.

Pacchetti modularizzati

La versione 2 dell'SDK for JavaScript (v2) richiedeva l'utilizzo dell'intero AWS SDK, come segue.

```
var AWS = require("aws-sdk");
```

Il caricamento dell'intero SDK non è un problema se l'applicazione utilizza molti servizi. AWS Tuttavia, se devi utilizzare solo alcuni AWS servizi, significa aumentare le dimensioni dell'applicazione con codice che non ti serve o che non usi.

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 nell'SDK for JavaScript, è necessario importare i pacchetti di AWS servizio completi, come illustrato nel codice seguente.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

Per chiamare i comandi v2 nel pattern async/await consigliato, utilizzate la seguente sintassi.

```
client.command(parameters);
```

L'esempio seguente utilizza il `createTable` comando v2 per creare una tabella DynamoDB utilizzando il pattern async/await consigliato.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
const 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 dell'SDK consentiva di modificare una richiesta durante le diverse fasi del suo ciclo di vita collegando listener di eventi alla richiesta. Questo approccio può rendere difficile il debug di ciò che è andato storto durante il ciclo di vita di una richiesta.

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 della v2 APIs 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 timestamp 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 client distorto.
  - v3: obsoleto. L'SDK applica 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 riferimento all'API 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. SDK inserisce sempre il prefisso del nome host quando necessario.
- `httpOptions`

  Un insieme di opzioni da passare alla richiesta HTTP di basso livello. 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 nel riferimento all'API v3 per NodeHttpHandler.

  Tutte le richieste v3 utilizzano HTTPS per impostazione predefinita. Devi solo fornire HttpAgent personalizzato.

```
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 fornire HttpAgent.

```
const { Agent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");

const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: new Agent({
            /*params*/
        }),
    }),
    endpoint: "http://example.com",
```

```
});
```

Se il client è in esecuzione nei browser, è disponibile un diverso set di opzioni. Puoi trovare ulteriori informazioni nella pagina di riferimento dell'API v3 per FetchHttpHandler.

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
const dynamodbClient = new DynamoDBClient({
    requestHandler: new FetchHttpHandler({
        requestTimeout: /* number in milliseconds */
    }),
});
```

Ciascuna opzione di `httpOptions` è specificata di seguito:

- `proxy`
  - v2: L'URL tramite il quale inoltrare le richieste tramite proxy.
  - 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 l'SDK invierà richieste HTTP asincrone.
  - v3: obsoleto. Le richieste sono sempre asincrone.
- `xhrWithCredentials`
  - v2: Imposta la proprietà «withCredentials» di un oggetto Request. XMLHttp
  - v3: non disponibile. SDK eredita 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. L'SDK non 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` Scopri di più nel riferimento all'API v3 per. RetryInputConfig Nota che `maxAttempts` dovrebbe esserlo`maxRetries + 1`.
- `paramValidation`
  - v2: Se i parametri di input devono essere convalidati rispetto alla descrizione dell'operazione prima di inviare la richiesta.
  - v3: obsoleto. SDK non 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. SDK supporta una strategia di riprova più flessibile con l'opzione Client Constructor. `retryStrategy` Scopri di più nella guida di riferimento all'API v3.
- `s3BucketEndpoint`
  - v2: Se l'endpoint fornito si rivolge a un singolo bucket (falso se si rivolge all'endpoint dell'API root).
  - v3: Modificato in. `bucketEndpoint` Scopri di più nel riferimento all'API v3 per BucketEndpoint. Tieni presente 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 della richiesta con la regione dedotta dall'ARN della risorsa richiesta.
  - 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 configurazione dell'API) è memorizzata nella cache.
  - v3: obsoleto. SDK memorizza sempre nella cache le chiavi di firma con hash.
- `signatureVersion`
  - v2: la versione di firma con cui firmare le richieste (sovrascrivendo la configurazione dell'API).
  - v3: obsoleto. La firma V2 supportata nell'SDK v2 è stata dichiarata obsoleta da. v3 supporta solo la firma v4. AWS
- `sslEnabled`
  - v2: Se SSL è abilitato per le richieste.
  - v3: Rinominato in. `tls`
- `stsRegionalEndpoints`
  - v2: se inviare la richiesta sts agli endpoint globali o agli endpoint regionali.
  - v3: obsoleto. Il client STS utilizzerà sempre gli endpoint regionali se impostato su una regione specifica. È possibile impostare la regione in cui inviare `aws-global` la richiesta all'endpoint globale STS.
- `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 delle EC2 istanze Amazon

  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. Inviaci una richiesta di funzionalità se ne hai bisogno.

```
// 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",
    },
  }),
});
```

EC2 Credenziali Amazon 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 da 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 Amazon 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: o `ECSCredentials` `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 strumenti. AWS SDKs 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. Comunemente usato in Amazon 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 APIs 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.
      array: [1, undefined], // The undefined value will be automatically omitted.
      map: { key: undefined }, // The "key" will be automatically omitted.
      set: new Set([1, undefined]), // The undefined 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

MD5 Checksum

Per saltare il calcolo dei MD5 checksum dei corpi dei messaggi, impostate su false nell'oggetto `md5` 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",
});
```

Documentazione supplementare

La tabella seguente include collegamenti alla documentazione supplementare che consente di utilizzare e comprendere il AWS SDK for JavaScript (v3).

Nome	Note
Client SDK	Informazioni sull'inizializzazione di un client SDK e parametri di costruzione configurabili comuni.
Aggiornamento delle note (da 2.x a 3.x)	Informazioni sull'aggiornamento da (v2). AWS SDK for JavaScript

Nome	Note
Utilizzo dei runtime AWS SDK for JavaScript (v3) su Node.js AWS Lambda	Procedure consigliate per AWS Lambda l'utilizzo di AWS SDK for JavaScript (v3).
Prestazioni	Informazioni su come il team AWS SDK ha ottimizzato le prestazioni dell'SDK e includono suggerimenti per configurare l'SDK in modo che funzioni in modo efficiente.
TypeScript	TypeScript suggerimenti e informazioni FAQs relative alla (v3). AWS SDK for JavaScript
Gestione degli errori	Suggerimenti per la gestione degli errori relativi a AWS SDK for JavaScript (v3).
Pratiche efficaci	Raccomandazioni generali per l'utilizzo di AWS SDK for JavaScript (v3).

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, è possibile iscriversi a un feed RSS.

Modifica	Descrizione	Data
TLS 1.3 è ora supportato su tutti gli endpoint delle API AWS di servizio in tutte le regioni	Versione TLS e metodo aggiornati per la registrazione della versione TLS.	10 aprile 2025
Generazione di client con @smithy /types	Contenuto aggiornato con la generazione di client utilizzan do il pacchetto @smithy / types.	15 febbraio 2025
Protezione dell'integrità dei dati con checksum	Contenuto aggiornato con dettagli sul calcolo automatico dei checksum.	15 gennaio 2025
Checksum di Amazon S3	È stata aggiunta una sezione su come utilizzare checksum flessibili con Amazon S3.	1 gennaio 2025
Support per endpoint basati su account in DynamoDB	Il supporto AWS SDK for JavaScript aggiunto per gli endpoint basati su account in DynamoDB.	26 settembre 2024
Nuovo argomento sulla registrazione dell'SDK	È stato aggiunto un argomento che descrive come registrare le chiamate API effettuate con	26 settembre 2024

	l'SDK for JavaScript , incluse informazioni sull'utilizzo del middleware per registrare le richieste.	
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 TLS minima	Sono state aggiunte informazioni su TLS 1.3.	31 marzo 2022
Impostare le 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 a usare Node.js for AWS SDK per 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
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