SDK Version3 のデベロッパーガイド

AWS SDK for JavaScript

AWS SDK for JavaScript: SDK Version3 のデベロッパーガイド

Table of Contents

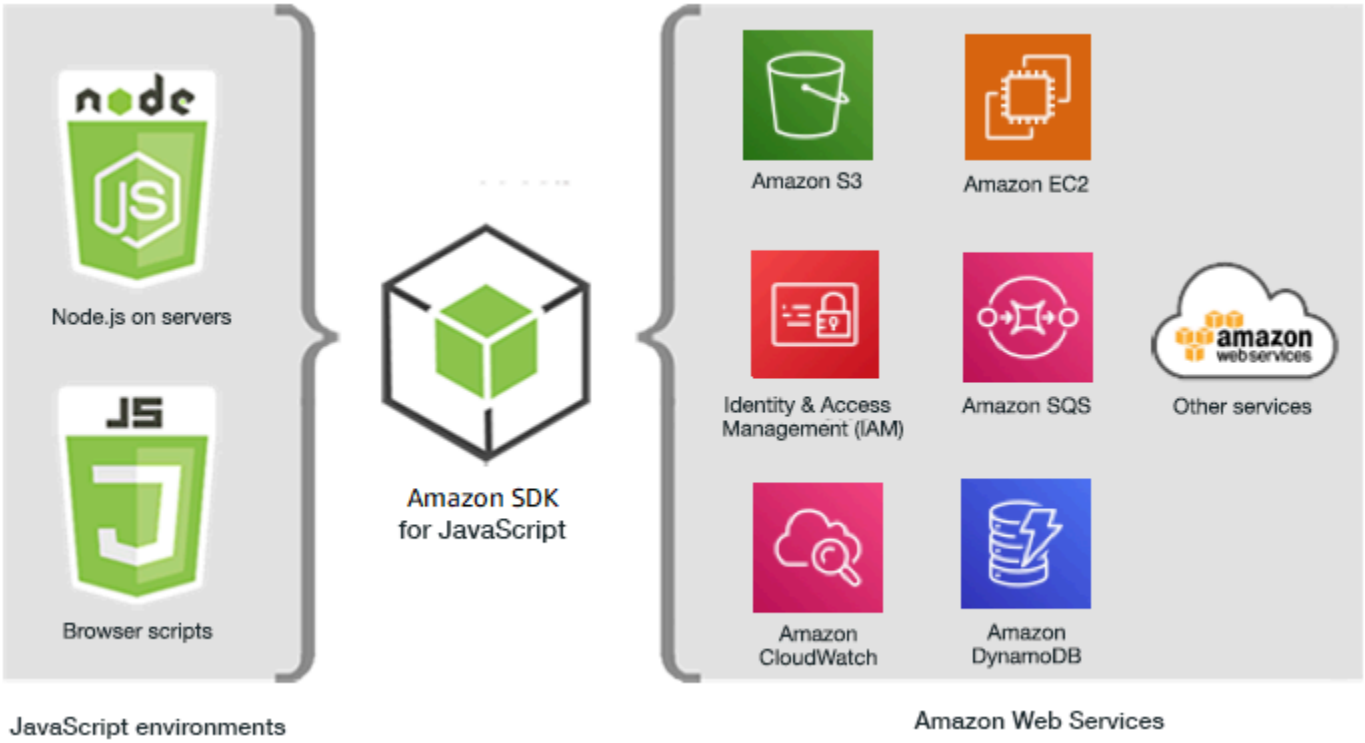
AWS SDK for JavaScript V3 API リファレンスガイドでは、 AWS SDK for JavaScript バージョン3 (V3) のすべての API オペレーションについて詳しく説明します。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。

とは AWS SDK for JavaScript

AWS SDK for JavaScript デベロッパーガイドへようこそ。このガイドは、 AWS SDK for JavaScript のセットアップおよび設定に関する一般的な情報を提供します。また、 を使用してさまざまな AWS サービスを実行する例とチュートリアルについても説明します AWS SDK for JavaScript。

AWS SDK for JavaScript v3 API Reference Guideは AWS のサービスのJavaScript APIを提供します。JavaScript API を使用して、Node.js またはブラウザ用のライブラリまたはアプリケーションを構築できます。



SDK の使用を開始する

SDK を使用する準備ができたら、「」の例に従いますはじめに。

開発環境を設定するには、「SDK for JavaScript のセットアップ」を参照してください。

現在 SDK for JavaScript のバージョン 2.x を使用している場合は、特定のガイダンスについて「v3 への移行」を参照してください。

のコード例をお探しの場合は AWS のサービス、「」を参照してくださいSDK for JavaScript (v3) のコード例。

SDK メジャーバージョンのメンテナンスとサポート

SDK メジャーバージョンのメンテナンスとサポート、およびその基礎的な依存関係については、AWS SDK とツール共有設定および認証情報リファレンスガイドで以下を参照してください。

- AWS SDKsメンテナンスポリシー
- AWS SDKsとツールのバージョンサポートマトリックス

Node.js で SDK を使用する

Node.js は、サーバー側の JavaScript アプリケーションを実行するための、クロスプラットフォームランタイムです。Node.js を Amazon Elastic Compute Cloud （ Amazon EC2 ）インスタンスで設定してサーバーで実行できます。Node.js を使用してオンデマンド AWS Lambda 関数を記述することもできます。

Node.js での SDK の使用方法は、ウェブブラウザの JavaScript で使用する方法とは異なります。この違いは、SDK のロード方法と、特定のウェブサービスにアクセスするために必要な認証情報の取得方法によるものです。特定の API の使用がNode.jsとブラウザの間で異なる場合、これらの違いを呼び出します。

での SDK の使用 AWS Amplify

ブラウザベースの Web、モバイル、ハイブリッドアプリケーションの場合は、 AWS Amplify library on GitHub を使用することもできます。JavaScript 用の SDK を拡張し、宣言タイプインターフェイスを提供します。

> **Note**
>
> Amplify などのフレームワークは、SDK for JavaScript と同じブラウザをサポートしない可能性があります。詳細については、フレームワークドキュメントを参照してください。

ウェブブラウザで SDK を使用します。

主要なウェブブラウザはすべて JavaScript の実行をサポートしています。ウェブブラウザで実行されている JavaScript コードは、クライアント側の JavaScript と呼ばれます。

でサポートされているブラウザのリストについては AWS SDK for JavaScript、「」を参照してください[サポートされているウェブブラウザ]。

ウェブブラウザでの SDK for JavaScript の使用方法は、Node.js を使用する方法とは異なります。この違いは、SDK のロード方法と、特定のウェブサービスにアクセスするために必要な認証情報の取得方法によるものです。特定の API の使用がNode.jsとブラウザの間で異なる場合、これらの違いを呼び出します。

V3 でブラウザーを使用

V3 を使用すると、必要なSDK for JavaScript ファイルのみをブラウザにバンドルして含めることができ、オーバーヘッドを削減できます。

SDK for JavaScriptのV3をHTMLページで使用するには、必要なクライアントモジュールと必要なすべてのJavaScript関数をWebpackを使用して単一のJavaScriptファイルにバンドルし、<head>HTMLページのスクリプトタグに追加する必要があります。例えば、

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

> **Note**
>
> Webpackの詳細については、[アプリケーションを webpack にバンドルする]を参照してください。

SDK for JavaScript の V2 を使用するには、代わりに V2 SDK の最新バージョンを指すスクリプトタグを追加します。詳細については、「 AWS SDK for JavaScript デベロッパーガイド v2」のサンプルを参照してください。

一般的なユースケース

ブラウザスクリプトで SDK for JavaScript を使用すると、多くの魅力的なユースケースを実現できます。SDK for JavaScript を使用してさまざまなウェブサービスにアクセスすることにより、ブラウザーアプリケーションで構築できるいくつかのアイデアを次に示します。

- カスタムコンソールを AWS サービスに構築します。このサービスでは、組織やプロジェクトのニーズに合わせて、リージョンやサービス全体の機能にアクセスして組み合わせます。

- Amazon Cognito アイデンティティを使用して、Facebook やその他のサードパーティーによる認証の使用を含めて、認証されたユーザーがブラウザアプリケーションやウェブサイトにアクセスできるようにします。
- Amazon Kinesis を使用して、クリックストリームやその他のマーケティングデータをリアルタイムで処理します。
- ウェブサイトの訪問者やアプリケーションユーザー向けの個別の優先ユーザー選定などの、サーバーレスデータの永続性のために Amazon DynamoDB を使用します。
- を使用して AWS Lambda 、知的財産をダウンロードしてユーザーに公開することなく、ブラウザスクリプトから呼び出すことができる独自のロジックをカプセル化します。

例について

AWS Code Example RepositoryでSKD for JavaScriptの例を参照できます。

リソース

このガイドに加えて、SDK for JavaScriptデベロッパーには次のオンラインリソースが利用可能です。

- AWS SDK for JavaScript V3 API リファレンスガイド
- AWS SDKsおよびツールリファレンスガイド: AWS SDKs。
- [JavaScript Developer Blog]
- AWS re:Post
- AWS コードライブラリの JavaScript の例
- AWS コードサンプルリポジトリ
- Gitter チャネル
- スタックオーバーフロー
- [Stack Overflow questions taggedAWS -sdk-js]
- GitHub
  - [SDK Source]
  - [Documentation Source]

の使用を開始する AWS SDK for JavaScript

AWS SDK for JavaScript は、ブラウザまたは Node.js 環境でウェブサービスへのアクセスを提供します。このセクションでは、こうしたそれぞれの JavaScript 環境で SDK for JavaScript を使用する方法を示す「使用開始」の演習が用意されています。

トピック

- を使用した SDK 認証 AWS
- Node.js での開始方法
- ブラウザでの開始方法
- React Nativeを開始します。

を使用した SDK 認証 AWS

を使用して開発 AWS するときに、コードが で認証される方法を確立する必要があります AWS のサービス。 AWS リソースへのプログラムによるアクセスは、環境と利用可能な AWS アクセスに応じてさまざまな方法で設定できます。

認証方法を選択して SDK 用に設定するには、AWS SDK とツールのリファレンスガイドの「認証とアクセス」を参照してください。

ローカルで開発中で、雇用主がセットアップするための認証方法を与えられていない新しいユーザーをお勧めします AWS IAM Identity Center。この方法には、設定を容易に AWS CLI するために をインストールしたり、 AWS アクセスポータルに定期的にサインインしたりすることが含まれます。この方法を選択した場合、AWS SDK とツールのリファレンスガイドの IAM Identity Center 認証の手順を完了したあと、環境には次の要素が含まれるはずです。

- アプリケーションを実行する前に AWS アクセスポータルセッションを開始 AWS CLIするために使用する 。
- SDK から参照できる設定値のセットを含む `[default]` プロファイルがある共有 AWSconfig ファイル。このファイルの場所を確認するには、AWS SDK とツールのリファレンスガイドの「共有ファイルの場所」を参照してください。
- 共有 `config` ファイルは `region` 設定を設定します。これにより、SDK AWS リージョン が AWS リクエストに使用するデフォルトが設定されます。このリージョンは、使用するリージョンが指定されていない SDK サービスリクエストに使用されます。

- SDK は、リクエストを AWSに送信する前に、プロファイルの SSO トークンプロバイダー設定を使用して認証情報を取得します。sso_role_name 値は、IAM Identity Center アクセス許可セットに接続された IAM ロールであり、アプリケーションで AWS のサービス 使用されている へのアクセスを許可します。

  次のサンプル config ファイルは、SSO トークンプロバイダー設定で設定されたデフォルトプロファイルを示しています。プロファイルの sso_session 設定は、指定された sso-session セクションを参照します。sso-session セクションには、 AWS アクセスポータルセッションを開始するための設定が含まれています。

```
[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
```

AWS SDK for JavaScript v3 では、IAM Identity Center 認証を使用するために、アプリケーションに追加のパッケージ (SSOや などSSOOIDC) は必要ありません。

この認証情報プロバイダーを明示的に使用する方法の詳細については、npm (Node.js パッケージマネージャー) ウェブサイトの fromSSO() を参照してください。

AWS アクセスポータルセッションを開始する

アクセスするアプリケーションを実行する前に AWS のサービス、SDK が IAM Identity Center 認証を使用して認証情報を解決するためのアクティブな AWS アクセスポータルセッションが必要です。設定したセッションの長さによっては、アクセスが最終的に期限切れになり、SDK で認証エラーが発生します。 AWS アクセスポータルにサインインするには、 で次のコマンドを実行します AWS CLI。

```
aws sso login
```

ガイダンスに従い、デフォルトのプロファイルを設定している場合は、`--profile` オプションを指定してコマンドを呼び出す必要はありません。SSO トークンプロバイダー設定で名前付きプロファイルを使用している場合、コマンドは `aws sso login --profile named-profile` です。

アクティブなセッションが既にあるかどうかをオプションでテストするには、次の AWS CLI コマンドを実行します。

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

セッションがアクティブな場合、このコマンドへの応答により、共有 `config` ファイルに設定されている IAM Identity Center アカウントとアクセス許可のセットが報告されます。

> **Note**
>
> 既にアクティブな AWS アクセスポータルセッションがあり、 を実行している場合は`aws sso login`、認証情報を指定する必要はありません。
> サインインプロセスにより、データ AWS CLI へのアクセスを許可するように求められる場合があります。 AWS CLI は SDK for Python 上に構築されているため、アクセス許可メッセージには`botocore`名前のバリエーションが含まれている可能性があります。

詳細認証情報

人間のユーザーとは、別名人間 ID と呼ばれ、人、管理者、デベロッパー、オペレーター、およびアプリケーションのコンシューマーを指します。 AWS 環境とアプリケーションにアクセスするには、ID が必要です。組織のメンバーである人間のユーザー、つまり、ユーザーや開発者は、ワークフォースアイデンティティと呼ばれます。

アクセス時に一時的な認証情報を使用します AWS。一時的な認証情報を提供するロールを引き受けることで、人間のユーザーの ID プロバイダーを使用して AWS アカウントへのフェデレーションアクセスを提供できます。一元的なアクセス管理を行うには、 AWS IAM Identity Center (IAM Identity Center) を使用して、ご自分のアカウントへのアクセスと、それらのアカウント内でのアクセス許可を管理することをお勧めします。その他の代替案については、以下を参照してください。

- ベストプラクティスの詳細については、IAM ユーザーガイドの「IAM でのセキュリティのベストプラクティス」を参照してください。
- 短期 AWS 認証情報を作成するには、IAM ユーザーガイドの「一時的なセキュリティ認証情報」を参照してください。

- 他の AWS SDK for JavaScript V3 認証情報プロバイダーの詳細については、「 SDK およびツールリファレンスガイド」の「標準化された認証情報プロバイダー」を参照してください。 AWS SDKs

Node.js での開始方法

このガイドでは、NPM パッケージを初期化し、パッケージにサービスクライアントを追加し、JavaScript SDK を使用してサービスアクションを呼び出す方法を説明します。

シナリオ

次の処理を実行するメインファイルを 1 つ含む、新しい NPM パッケージを作成します。

- Amazon Simple Storage Service バケットの作成
- Amazon S3 バケットへのオブジェクトの配置
- Amazon S3 バケット内のオブジェクトの読み取り
- ユーザーがリソースを削除したいかどうかの確認

前提条件

例を実行するには、次の手順を行います。

- SDK 認証を設定します。詳細については、「を使用した SDK 認証 AWS」を参照してください。
- 開発用の Node.js のアクティブ LTS バージョンを使用して Node.js. AWS recommends をインストールします。

ステップ 1: パッケージ構造を設定してクライアントパッケージをインストールする

パッケージ構造を設定し、クライアントパッケージをインストールするには、次の手順を実行します。

1. 新しいフォルダ `nodegetstarted` を作成して、パッケージを格納します。
2. コマンドラインから、新しいフォルダに移動します。
3. 次のコマンドを実行して、デフォルト `package.json` ファイルを作成します。

```
npm init -y
```

4. 次のコマンドを実行して、Amazon S3 クライアントパッケージをインストールします。

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

5. "type": "module" を package.json ファイルに追加します。これにより、最新の ESM 構文を使用するように Node.js に指示します。最終的な package.json は次のようになります。

```
{
  "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"
}
```

ステップ 2: 必要なインポートと SDK コードを追加する

nodegetstarted フォルダー内の index.js という名前のファイルに、次のコードを追加します。

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

サンプルコードは、このGitHubで見つけることができます。

ステップ 3: 例を実行する

> **Note**
>
> 必ずサインインしてください。IAM Identity Center を使用して認証する場合は、 コマンドを使用して AWS CLI `aws sso login`サインインすることを忘れないでください。

1. `node index.js` を実行します。
2. バケットを空にして削除するかどうかを選択します。
3. バケットを削除しない場合は、手動で空にして後で削除してください。

ブラウザでの開始方法

このセクションでは、ブラウザでSDK for JavaScriptのバージョン3(V3)を実行する方法を示す例を順番に説明します。

> **Note**
>
> ブラウザでV3を実行することは、バージョン2 (V2)とは若干異なります。詳細については、「V3 でブラウザーを使用」を参照してください。

SDK for JavaScript (V3) を使用するその他の例については、SDK for JavaScript (v3) のコード例 を参照してください。

このウェブアプリケーションの例は、以下を示します。

- 認証に Amazon Cognito を使用して AWS サービスにアクセスする方法。
- AWS Identity and Access Management (IAM) ロールを使用して Amazon Simple Storage Service (Amazon S3) バケット内のオブジェクトのリストを読み取る方法。

> **Note**
>
> この例では、認証 AWS IAM Identity Center に を使用しません。

シナリオ

Amazon S3 は、業界をリードするスケーラビリティ、データ可用性、セキュリティ、パフォーマンスを提供するオブジェクトストレージサービスです。Amazon S3 を使用して、バケットと呼ばれるコンテナ内にデータをオブジェクトとして保存できます。Amazon S3 の詳細については、「Amazon S3 ユーザーガイド」を参照してください。

この例では、Amazon S3 バケットから読み取る IAM ロールを引き受けるウェブアプリケーションを設定して実行する方法を示します。この例では、React フロントエンドライブラリと Vite フロントエンドツールを使用して JavaScript 開発環境を提供します。ウェブアプリは Amazon Cognito ID プールを使用して、 AWS サービスへのアクセスに必要な認証情報を提供します。含まれているコードサンプルの例は、ウェブアプリケーションでSDK for JavaScriptをロードして使用するための基本的なパターンを示します。

ステップ 1: Amazon Cognito アイデンティティプールと IAM ロールを作成する

この演習では、Amazon Cognito アイデンティティプールを作成して使用し、Amazon S3 サービスのウェブアプリケーションで未認証のアクセスを提供します。ID プールを作成すると、認証されていないゲストユーザーをサポートする AWS Identity and Access Management (IAM) ロールも作成されます。この例では、タスクに集中し続けるために、認証されていないユーザーロールのみを使用します。後で ID プロバイダーと認証済みユーザーのサポートを統合できます。Amazon Cognito アイデンティティプールの追加についての詳細は、「Amazon Cognito デベロッパーガイド」の「チュートリアル: ID プールの作成」を参照してください。

Amazon Cognito アイデンティティプールおよび関連付けられた IAM ロールを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cognito/ で Amazon Cognito コンソールを開きます。
2. 左のナビゲーションペインで、[ID プール] を選択します。
3. [ID プールを作成] を選択します。
4. [ID プールの信頼を設定] で、ユーザー認証に [ゲストアクセス] を選択します。
5. [アクセス許可を設定] で、[新しい IAM ロールの作成] を選択し、[IAM ロール名] に名前 (getStartedRole など) を入力します。
6. [プロパティを設定] で、[ID プール名] に名前 (getStartedPool など) を入力します。

7. [確認および作成] で、新しいアイデンティティプールに対して行った選択を確認します。[編集] を選択してウィザードに戻り、設定を変更します。終了したら、[ID プールの作成] を選択します。
8. [ID プールの ID] と、新しく作成した Amazon Cognito アイデンティティプールの [リージョン] を書き留めます。ステップ 4: ブラウザコードを設定する で *IDENTITY_POOL_ID* および *REGION* を置換するには、これらの値が必要です。

Amazon Cognito アイデンティティプールを作成したら、ウェブアプリケーションにより必要な Amazon S3 権限を追加する準備が整います。

ステップ 2: 作成した IAM ロールにポリシーを追加する

ウェブアプリケーション内の Amazon S3 バケットへのアクセスを有効にするには、Amazon Cognito アイデンティティプール (getStartedPool など) 用に作成された非認証の IAM ロール (getStartedRole など) を使用します。これを進めるには、IAM ポリシーをロールにアタッチする必要があります。IAM ロールの変更の詳細については、「IAM ユーザーガイド」の「ロールのアクセス許可ポリシーの変更」を参照してください。

Amazon S3ポリシーを、非認証ユーザーに関連付けられているIAM ロールに追加するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. 変更するロールの名前 (getStartedRole など) を選択し、[アクセス許可] タブを選択します。
4. [アクセス許可を追加]、[ポリシーをアタッチ] の順に選択します。
5. このロールの [アクセス許可を追加] ページで、AmazonS3ReadOnlyAccess を検索してチェックボックスをオンにします。

> **Note**
>
> このプロセスを使用して、任意の AWS サービスへのアクセスを有効にできます。

6. [Add permissions (許可の追加)] を選択します。

Amazon Cognito アイデンティティプールを作成した後、非認証ユーザーの IAM ロールに Amazon S3の許可を追加すると、Amazon S3 バケットを追加し設定する準備が整います。

ステップ 3: Amazon S3 バケットとオブジェクトを追加する

このステップでは、例の Amazon S3 バケットとオブジェクトを追加します。また、バケットの Cross-Origin Resource Sharing (CORS) を有効にします。Amazon S3 バケットとオブジェクトの作成についての詳細は、「Amazon S3 ユーザーガイド」の「Amazon S3 の開始方法」を参照してください。

CORS でAmazon S3 バケットとオブジェクトを追加するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. 左側のナビゲーションペインで、[バケット] を選択してから、[バケットを作成] を選択します。
3. バケットの命名規則 (getstartedbucket など) に準拠したバケット名を入力し、[バケットを作成] を選択します。
4. 作成したバケットを選択し、[オブジェクト] タブを選択します。次に、アップロードを選択します。
5. [Files and Folders (ファイルとフォルダ)] で、[Add files (ファイルを追加)] を選択します。
6. アップロードするファイルを選択し、続いて [オープン] を選択します。次に、[アップロード] を選択して、バケットへのオブジェクトのアップロードを完了します。
7. 次に、バケットの [アクセス許可] タブを選択し、[Cross-Origin Resource Sharing (CORS)] セクションで [編集] を選択します。次の JSON を入力します。

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

8. [Save changes] (変更の保存) をクリックします。

Amazon S3 バケットを追加してオブジェクトを追加したら、ブラウザコードを設定する準備が整います。

ステップ 4: ブラウザコードを設定する

サンプルアプリケーションは単一ページの React アプリケーションで構成されています。この例のファイルは、この GitHub で見つけることができます。

サンプルアプリケーションを設定するには

1. Node.js をインストールします。
2. コマンドラインから、AWS のコードサンプルリポジトリをクローンします。

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

3. サンプルアプリケーションに移動します。

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

4. 次のコマンドを実行して、必要なパッケージをインストールします。

```
npm install
```

5. 次に、テキストエディタで `src/App.tsx` を開き、次の処理を実行します。

   - *YOUR_IDENTITY_POOL_ID* を ステップ 1: Amazon Cognito アイデンティティプールと IAM ロールを作成する で書き留めた Amazon Cognito アイデンティティプール ID に置き換えます。
   - リージョンの値を Amazon S3 バケットと Amazon Cognito アイデンティティプールに割り当てられたリージョンに置き換えます。両方のサービスのリージョンは同じでなければならないことに注意してください (us-east-2 など)。
   - *bucket-name* を、ステップ 3: Amazon S3 バケットとオブジェクトを追加する で作成したバケットの名前に置き換えます。

テキストを置き換えたら、`App.tsx` ファイルを保存します。これで、ウェブアプリケーションを実行する準備ができました。

ステップ 5: 例を実行する

サンプルアプリケーションを実行するには

1. コマンドラインから、サンプルアプリケーションに移動します。

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

2. コマンドラインから、以下のコマンドを実行します。

```
npm run dev
```

Vite 開発環境が実行され、次のメッセージが表示されます。

```
VITE v4.3.9  ready in 280 ms

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

3. ウェブブラウザで上記の URL (例: http://localhost:5173) に移動します。サンプルアプリケーションでは、Amazon S3 バケット内のオブジェクトファイル名のリストが表示されます。

クリーンアップ

このチュートリアルで作成されたリソースをクリーンアップするには、以下を行います。

- Amazon S3 コンソールで、作成されたオブジェクトとバケット (getstartedbucket など) をすべて削除します。
- IAM コンソールで、ロール名 (getStartedRole など) を削除します。
- Amazon Cognito コンソールでアイデンティティプール名 (getStartedPool など) を削除します。

React Nativeを開始します。

このチュートリアルでは、ReactNative CLIを使用して React Native アプリを作成する方法を示します。

このチュートリアルでは、次のことを示します。

- プロジェクトが使用する AWS SDK for JavaScript バージョン 3 (V3) モジュールをインストールして含める方法。
- 「Amazon Simple Storage Service (Amazon S3)」に接続して Amazon S3 バケットを作成および削除するコードを記述する方法。

シナリオ

Amazon S3は、ウェブ上のどこからでも、いつでも、任意の量のデータを保存および取得できるようにするクラウドサービスです。React Native は、モバイルアプリケーションを作成して有効にするデベロップメントフレームワークです。このチュートリアルでは、AmazonS3に接続してAmazonS3バケットを作成および削除するReactNativeアプリを作成する方法を示します。

アプリケーションは 次のSDK for JavaScript API を使用します。

- `CognitoIdentityClient` コンストラクタ
- `S3` コンストラクタ

前提条件タスク

> **Note**
>
> 他のチュートリアルまたは既存の設定で次のステップのいずれかをすでに完了している場合は、これらのステップをスキップします。

このセクションでは、このチュートリアルを完了するために必要な最小限のセットアップを提供します。これを完全なセットアップであるとは見なさないでください。詳細については、「SDK for JavaScript のセットアップ」を参照してください。

- 次のツールをインストールします。

  - npm
  - Node.js
  - iOS でテストしている場合は Xcode
  - Android でテストしている場合は Android Studio
- React Native 開発環境をセットアップする
- これらの Node TypeScript の例を実行するようにプロジェクト環境を設定し、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- AWS サービスを使用して開発 AWS するときに、コードが で認証される方法を確立する必要があります。詳細については、「を使用した SDK 認証 AWS」を参照してください。

> **Note**
>
> この例の IAM ロールは、AmazonS3FullAccess アクセス許可を使用するように設定する必要があります。

ステップ 1: Amazon Cognito アイデンティティプールを作成

この演習では、Amazon Cognito アイデンティティプールを作成して使用し、Amazon S3 サービスのアプリで未認証のアクセスを提供します。ID プールを作成すると、2 つの AWS Identity and Access Management (IAM) ロールも作成されます。1 つは ID プロバイダーによって認証されたユーザーをサポートするロール、もう 1 つは認証されていないゲストユーザーをサポートするロールです。

この演習では、タスクに集中し続けるために、認証されていないユーザーロールのみを使用します。後で ID プロバイダーと認証済みユーザーのサポートを統合できます。

Amazon Cognito アイデンティティプールを作成するには

1. にサインイン AWS Management Console し、Amazon Web Services Console で Amazon Cognito コンソールを開きます。
2. コンソールの開きページで ID プールを選択します。
3. 次のページで、[新しい ID プールの作成] を選択します。

> **Note**
>
> 他のアイデンティティプールがない場合、Amazon Cognito コンソールはこのページをスキップし、代わりに次のページを開きます。

4. [ID プールの信頼を設定] で、ユーザー認証に [ゲストアクセス] を選択します。
5. アクセス許可の設定 で、新しい IAM ロールの作成 を選択し、IAM ロール名に名前 (getStartedReactRole など) を入力します。
6. Configure プロパティで、ID プール名に名前 (getStartedReactPool など) を入力します。
7. [確認および作成] で、新しいアイデンティティプールに対して行った選択を確認します。[編集] を選択してウィザードに戻り、設定を変更します。終了したら、[ID プールの作成] を選択します。
8. この新しく作成された ID プールの ID とリージョンを書き留めます。ブラウザスクリプトで *region* と *identityPoolId* を置き換えるには、これらの値が必要です。

Amazon Cognito アイデンティティプールを作成したら、React Native appにより必要な Amazon S3 権限を追加する準備が整います。

ステップ 2: 作成した IAM ロールにポリシーを追加

Amazon S3 へのブラウザスクリプトのアクセスを有効にして Amazon S3 バケットを作成および削除するには、Amazon Cognito ID プール用に作成された非認証の IAM ロールを使用します。これを進めるには、IAM ポリシーをロールに追加する必要があります。IAM ロールの詳細については、IAM ユーザーガイドの「AWS サービスにアクセス許可を委任するロールの作成」を参照してください。

Amazon S3ポリシーを、非認証ユーザーに関連付けられているIAM ロールに追加するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. 変更するロールの名前 (getStartedRole など) を選択し、[アクセス許可] タブを選択します。
4. [アクセス許可を追加]、[ポリシーをアタッチ] の順に選択します。
5. このロールの [アクセス許可を追加] ページで、AmazonS3ReadOnlyAccess を検索してチェックボックスをオンにします。

> **Note**
>
> このプロセスを使用して、任意の AWS サービスへのアクセスを有効にできます。

6. [Add permissions (許可の追加)] を選択します。

Amazon Cognito アイデンティティプールを作成した後、非認証ユーザーの IAM ロールに Amazon S3の許可を追加すると、アプリを構築する準備が整います。

ステップ 3: Create-react-native-app を使用してアプリケーションを作成します

次のコマンドを実行してReact Native Appを作成します。

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

ステップ 4: Amazon S3パッケージとその他の依存関係をインストールします

プロジェクトのディレクトリ内で、次のコマンドを実行して Amazon S3 パッケージをインストールします。

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

このコマンドはプロジェクトに Amazon S3 パッケージをインストールして、`package.json`を更新し、 Amazon S3 をプロジェクトの依存関係として一覧表示します。このパッケージに関する情報は、https://www.npmjs.com/npmウェブサイトで「@aws-sdk」と検索すると見つかります。

これらのパッケージとそれに関連するコードは、プロジェクトの `node_modules` サブディレクトリにインストールされています。

Node.js パッケージのインストールの詳細については、npm (Node.js パッケージマネージャー) ウェブサイトのパッケージをローカルでダウンロードしてインストールおよび Node.jsモジュールの作成を参照してください。のダウンロードとインストールの詳細については AWS SDK for JavaScript、「」を参照してくださいSDK for JavaScript をインストールする。

認証に必要なその他の依存関係をインストールします。

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

ステップ 5: React Native コードを書き込みする。

次のコードをApp.tsxに追加します。*identityPoolId* と *region* を、Amazon S3 バケットが作成される ID プール ID と Region に置き換えます。

```
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;
```

コードはまず、必要な React、React Native、および AWS SDK の依存関係をインポートします。

機能アプリ内部：

- S3Client オブジェクトが作成され、前に作成した Amazon Cognito アイデンティティプールを使用して認証情報を指定します。
- 方法`createBucket`と`deleteBucket`で指定されたバケットをそれぞれ作成および削除します。
- React ネイティブビューには、ユーザーが Amazon S3 バケット名を指定するテキスト入力フィールドと、指定された Amazon S3 バケットを作成および削除するボタンが表示されます。

JavaScriptの全ページはこのGitHubで公開されています。

ステップ6。例を実行する

> **Note**
>
> 必ずサインインしてください。IAM Identity Center を使用して認証する場合は、 コマンドを使用して AWS CLI `aws sso login`サインインすることを忘れないでください。

この例を実行するには、npm を使用して `web`、`ios`、または `android` コマンドを実行します。

これは、macOSで実行中の`ios`コマンドの出力例です。

```
$ 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
```

これは、macOSで実行中の`android`コマンドの出力例です。

```
$ 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 }
```

作成または削除したいバケット名を入力し、Create Bucket (バケットの作成) またはDelete Bucket (バケットの削除) のいずれかをクリックします。それぞれのコマンドが Amazon S3 に送信され、成功またはエラーメッセージが表示されます。

想定される拡張機能

これは、React Native app のSDK for JavaScript を使用してさらに検索するために使用できる、このアプリケーションのバリエーションを示します。

- Amazon S3 バケットを一覧表示するボタンを追加し、一覧にされている各バケットの横に削除ボタンを提供します。
- テキストオブジェクトをバケットに入れるためのボタンを追加します。
- 認証済みの IAM ロールを使用するために、Facebook や Amazon などの外部のアイデンティティプロバイダーと統合します。

SDK for JavaScript のセットアップ

このセクションのトピックでは、SDK for JavaScript をインストールおよびロードし、SDK がサポートするウェブサービスにアクセスする方法について説明します。

トピック

前提条件

開発用の Node.js のアクティブ LTS バージョンを使用して Node.js. AWS recommends をインストールします。

トピック

AWS Node.js 環境をセットアップする

アプリケーションを実行できる AWS Node.js 環境を設定するには、次のいずれかの方法を使用します。

- Node.js がプリインストールされている Amazon マシンイメージ (AMI) を選択します。次に、その AMI を使用してAmazon EC2 インスタンスを作成します。Amazon EC2 インスタンスを作成するときは、 AWS Marketplaceから AMI を選択してください。 AWS Marketplace で Node.js を検索し、Node.js のプリインストールされたバージョン (32 ビットまたは 64 ビット) を含む AMI オプションを選択します。
- Amazon EC2 インスタンスを作成して、Node.js をインストールします。Amazon Linux インスタンスで Node.js をインストールする方法の詳細については、Amazon EC2 インスタンスでの Node.js を設定するを参照してください。
- を使用してサーバーレス環境を作成し AWS Lambda 、Lambda 関数として Node.js を実行します。Lambda 関数内で Node.js を使用する方法の詳細は、[AWS Lambda Developer Guide]の[Programming model (Node.js)]を参照してください。

- Node.js アプリケーションを にデプロイします AWS Elastic Beanstalk。Elastic Beanstalk で Node.js を使用する方法の詳細については、AWS Elastic Beanstalk [Developer Guide]の[Deploying Node.js applications to AWS Elastic Beanstalk]を参照してください。
- を使用して Node.js アプリケーションサーバーを作成します AWS OpsWorks。で Node.js を使用する方法の詳細については OpsWorks、 AWS OpsWorks ユーザーガイドの「最初の Node.js スタックの作成」を参照してください。

サポートされているウェブブラウザ

は、最新のすべてのウェブブラウザ AWS SDK for JavaScript をサポートしています。

バージョン 3.567.0 以降では、SDK for JavaScript は次の最小バージョンをサポートする ES2021 アーティファクトを出力します。

ブラウザ	バージョン
Google Chrome	85.0 以降
Mozilla Firefox	80.0 以降
Opera	71.0 以降
Microsoft Edge	85.0 以降
Apple Safari	14.1 以降
サムスン・インターネット	14.0 以降

バージョン 3.183.0 から 3.566.0 では、SDK for JavaScript は次の最小バージョンをサポートする ES2020 アーティファクトを使用します。

ブラウザ	バージョン
Google Chrome	80.0 以降
Mozilla Firefox	80.0 以降
Opera	63.0 以降

ブラウザ	バージョン
Microsoft Edge	80.0 以降
Apple Safari	14.1 以降
サムスン・インターネット	12.0 以降

バージョン 3.182.0 以前は、SDK for JavaScript は ES5 アーティファクトを使用します。このアーティファクトは、以下の最小バージョンをサポートしています。

ブラウザ	バージョン
Google Chrome	49.0 以降
Mozilla Firefox	45.0 以降
Opera	36.0 以降
Microsoft Edge	12.0 以降
Windows Internet Explorer	該当なし
Apple Safari	9.0 以降
Android ブラウザ	76.0 以降
UC ブラウザ	12.12 以降
サムスン・インターネット	5.0 以降

> **Note**
>
> などのフレームワークは、SDK for JavaScript と同じブラウザサポートを提供しない AWS Amplify 場合があります。詳細については、[AWS Amplify Documentation]を参照してください。

SDK for JavaScript をインストールする

すべてのサービスが SDK またはすべての AWS リージョンですぐに利用できるわけではありません。

Node.js パッケージマネージャーである npm AWS SDK for JavaScript を使用して からサービスをインストールするには、コマンドプロンプトで次のコマンドを入力します。### SERVICE は などのサービスの名前ですs3。

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

AWS SDK for JavaScript サービスクライアントパッケージの完全なリストについては、 AWS SDK for JavaScript API リファレンスガイドを参照してください。

SDK for JavaScript をロードする

SDK のインストール後、import を使用してノードアプリケーションにクライアントパッケージをロードできます。たとえば、Amazon S3 クライアントと Amazon S3 ListBuckets コマンドをロードするには、以下を使用します。

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

SDK for JavaScript を設定する

SDK for JavaScript を使用して、API を使用するウェブサービスを呼び出す前に、SDK を設定する必要があります。少なくとも、以下を設定する必要があります。

- サービスをリクエストする AWS リージョン
- コードが で認証される方法 AWS

これらの設定に加えて、 AWS リソースへの許可も設定する必要があります。例えば、Amazon S3 バケットへのアクセスを制限したり、Amazon DynamoDB テーブルを読み取り専用アクセスに制限したりできます。

AWS SDKs およびツールリファレンスガイドには、多くの AWS SDKs で一般的な設定、機能、その他の基本的な概念も含まれています。

このセクションのトピックでは、WebブラウザーでNode.jsおよびJavaScript用のSDK for JavaScriptを設定する方法について説明します。

トピック

サービスごとの設定

SDK を設定するには、設定情報をサービスオブジェクトに渡します。

サービスレベルの構成では、個々のサービスを大幅に制御し、ニーズがデフォルトの設定と異なる場合に、個々のサービスオブジェクト設定の更新を有効にします。

> **Note**
>
> バージョン 2.x では、 AWS SDK for JavaScript サービス設定を個々のクライアントコンストラクタに渡すことができます。ただし、これらの設定は、まずグローバル SDK 設定 `AWS.config`のコピーに自動的にマージされます。

また、既存のクライアントではなく、更新呼び出しが行われた後にインスタンス化されたサービスクライアントのAWS.config.update({/* params */})更新された設定のみを呼び出します。
この動作は頻繁な混乱の原因であり、フォワード互換の方法でサービスクライアントのサブセットにのみ影響する設定をグローバルオブジェクトに追加することを困難にしました。
バージョン3では、SDKによって管理されるグローバル設定はなくなりました。設定は、インスタンス化された各サービスクライアントに渡す必要があります。同じ設定を複数のクライアント間で共有することは可能ですが、その設定はグローバルステートに自動的にマージされません。

サービスごとに設定を設定する

SDK for JavaScript で使用する各サービスには、そのサービスの API の一部であるサービスオブジェクトを介してアクセスします。例えば、Amazon S3 サービスにアクセスするには、Amazon S3 サービスオブジェクトを作成します。そのサービスオブジェクトのコンストラクタの一部として、サービスに固有の設定を指定できます。

例えば、複数の AWS リージョンで Amazon EC2 オブジェクトにアクセスする必要がある場合は、リージョンごとに Amazon EC2 サービスオブジェクトを作成し、それに応じて各サービスオブジェクトのリージョン設定を設定します。

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

AWS リージョンを設定する

AWS リージョンは、同じ地理的領域にある AWS リソースの名前付きセットです。リージョンの例は us-east-1 です。これは、米国東部 (バージニア北部) リージョンです。SDK がその地域内のサービスにアクセスするように、SDK for JavaScript でサービスクライアントを作成するときに地域を指定します。 の一部のサービスは、特定のリージョンでのみ利用可能です。

SDK for JavaScript は、デフォルトではリージョンを選択しません。ただし、環境変数または共有設定configファイルを使用して AWS リージョンを設定できます。

クライアントクラスコンストラクタ内

サービスオブジェクトをインスタンス化するときは、次に示すように、クライアントクラスコンストラクタの一部としてそのリソースの AWS リージョンを指定できます。

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

環境変数を使用する

AWS_REGION 環境変数を設定して、リージョンを設定できます。この変数を定義すると、SDK for JavaScript がそれを読み込み、使用します。

共有設定ファイルを使用する

共有認証情報ファイルで SDK で使用する認証情報を保存できるのと同様に、 AWS リージョンやその他の設定は、SDK config が使用する という名前の共有ファイルに保持できます。AWS_SDK_LOAD_CONFIG 環境変数が真の値に設定されている場合、SDK for JavaScript はロード時にconfigファイルを自動的に検索します。config ファイルを保存する場所はオペレーティングシステムによって異なります。

- Linux、macOS、またはUnix ユーザー - ~/.aws/config
- Windows ユーザー - C:\Users\USER_NAME\.aws\config

共有 config ファイルがまだない場合は、指定されたディレクトリに 1 つ作成することができます。次の例では、config ファイルはリージョンと出力形式の両方を設定します。

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

共有された config および credentials ファイルの使用についての詳細は、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

地域設定の優先順位

地域設定の優先順位は以下の通りです。

1. リージョンがクライアントクラスコンストラクタに渡された場合、そのリージョンが使用されます。

2. 環境変数に地域が設定されている場合は、その地域が使用されます。

3. それ以外の場合は、共有 config ファイルで定義された地域が使用されます。

認証情報の設定

AWS は認証情報を使用して、サービスを呼び出すユーザーと、リクエストされたリソースへのアクセスが許可されているかどうかを識別します。

ウェブブラウザでも、Node.js サーバーでも、JavaScript コードは API を介してサービスにアクセスする前に有効な認証情報を取得する必要があります。認証情報は、認証情報をサービスオブジェクトに直接渡してサービスごとに設定することができます。

ウェブブラウザで Node.js と JavaScript の間で異なる認証情報を設定する方法はいくつかあります。このセクションのトピックでは、Node.js またはウェブブラウザで認証情報を設定する方法について説明します。いずれの場合も、オプションは推奨順に表示されています。

認証情報のベストプラクティス

認証情報を正しく設定することで、ミッションクリティカルなアプリケーションに影響を与えたり重要なデータを侵害する可能性があるセキュリティ問題への露出を最小限に抑えながら、アプリケーションまたはブラウザスクリプトが必要なサービスおよびリソースにアクセスできるようにします。

認証情報を設定するときに適用する重要な原則は、常に自分のタスクに必要な最小限の権限を付与することです。最小限のアクセス許可を超えるアクセス許可を提供し、その結果、セキュリティ問題が後で発見されてそれを修正するよりも、リソースに対する最小限のアクセス許可を提供し、必要に応じてさらにアクセス許可を追加する方が安全です。例えば、Amazon S3 バケット内のオブジェクトや DynamoDB テーブル内のオブジェクトなど、個々のリソースを読み書きする必要がある場合を除き、これらのアクセス許可を読み取り専用に設定します。

最小特権の付与の詳細については、「IAM ユーザーガイド」のベストプラクティストピックの「最小特権を付与」を参照してください。

トピック

- Node.js で認証情報を設定する
- ウェブブラウザで認証情報を設定する

Node.js で認証情報を設定する

ローカルで開発中で、雇用主から認証方法を与えられていない新規ユーザーは、セットアップすることをお勧めします AWS IAM Identity Center。詳細については、「を使用した SDK 認証 AWS」を参照してください。

Node.js では、SDK に認証情報を提供する方法がいくつかあります。これらの中には、より安全なものもあれば、アプリケーションの開発中により便利に使えるものもあります。Node.js で認証情報を取得する場合は、環境変数やロードした JSON ファイルなど、1つ以上のソースに依存するように注意してください。変更が行われたことに気付かずに、コードの実行に使用されるアクセス許可を変更してしまう可能性があります。

AWS SDK for JavaScript V3 は Node.js でデフォルトの認証情報プロバイダーチェーンを提供するため、認証情報プロバイダーを明示的に指定する必要はありません。デフォルトの認証情報プロバイダーチェーンは、認証情報が 1 つのソースから返されるまで指定された優先順位で、さまざまな異なるソースからの認証情報を解決しようとします。SDK for JavaScript V3 の認証情報プロバイダーチェーンについては、こちらを参照してください。

認証情報プロバイダーチェーン

すべての SDK には、 AWS のサービスに対するリクエストに使用する有効な認証情報を取得するためにチェックする一連の場所 (またはソース) があります。有効な認証情報が見つかると、検索は停止されます。この体系的な検索は、デフォルトの認証情報プロバイダーチェーンと呼ばれます。

チェーンのステップごとに、値を設定するさまざまな方法があります。コードで直接値を設定することが常に優先され、次に環境変数として を設定し、次に共有 AWS `config`ファイルで を設定します。詳細については、『AWS SDK とツールのリファレンスガイド』の「設定の優先順位」を参照してください。

AWS SDKsおよびツールリファレンスガイドには、すべての SDK および で使用される AWS SDKs 設定に関する情報が記載されています AWS CLI。共有 AWS `config`ファイルを使用して SDK を設定する方法の詳細については、「共有設定ファイルと認証情報ファイル」を参照してください。環境変数を設定して SDK を設定する方法の詳細については、「環境変数のサポート」を参照してください。

で認証するために AWS、 は、次の表に示す順序で認証情報プロバイダー AWS SDK for JavaScript をチェックします。

AWS SDK for JavaScript API 優先順位による認証情報プロバイダーメソッドの参照	使用可能な認証情報プロバイダー	AWS SDKsとツールのリファレンスガイド
`fromEnv()`	AWS 環境変数からのアクセスキー	AWS アクセスキー
`fromSSO()`	AWS IAM Identity Center。このガイドでは、を使用した SDK 認証 AWS を参照してください。	IAM Identity Center 認証情報プロバイダー
`fromIni()`	AWS 共有ファイル`config`と`credentials` ファイルからのアクセスキー	AWS アクセスキー
	信頼されたエンティティプロバイダー (`AWS_ROLE_ARN` など)	IAM ロールの継承
	AWS Security Token Service (AWS STS) からのウェブ ID トークン	ウェブアイデンティティまたは OpenID Connect でのフェデレーション
	Amazon Elastic Container Service (Amazon ECS) 認証情報	コンテナ認証情報プロバイダー
	Amazon Elastic Compute Cloud (Amazon EC2) インスタンスプロファイル認証情報 (IMDS 認証情報プロバイダー)	IMDS 認証情報プロバイダー
	プロセス認証情報プロバイダー	プロセス認証情報プロバイダー
	AWS IAM Identity Center 認証情報	IAM Identity Center 認証情報プロバイダー

AWS SDK for JavaScript API 優先順位による認証情報プロバイダーメソッドの参照	使用可能な認証情報プロバイダー	AWS SDKsとツールのリファレンスガイド
fromProcess()	プロセス認証情報プロバイダー	プロセス認証情報プロバイダー
fromTokenFile()	AWS Security Token Service (AWS STS) からのウェブ ID トークン	ウェブアイデンティティまたは OpenID Connect でのフェデレーション
fromContainerMetadata()	Amazon Elastic Container Service (Amazon ECS) 認証情報	コンテナ認証情報プロバイダー
fromInstanceMetadata()	Amazon Elastic Compute Cloud (Amazon EC2) インスタンスプロファイル認証情報 (IMDS 認証情報プロバイダー)	IMDS 認証情報プロバイダー

新規ユーザーに推奨されるアプローチに従って開始した場合は、「使用開始」のトピックの を使用した SDK 認証 AWS 中に AWS IAM Identity Center 認証を設定します。その他の認証方法もさまざまな状況で役に立ちます。セキュリティリスクを避けるため、常に短期の認証情報を使用することをお勧めします。その他の認証方法については、「AWS SDK とツールのリファレンスガイド」の「認証とアクセス」を参照してください。

このセクションのトピックでは、認証情報を Node.js にロードする方法について説明します。

トピック

- Amazon EC2 の IAM ロールから Node.js に認証情報をロードする
- Node.js Lambda 関数の認証情報をロードする

Amazon EC2 の IAM ロールから Node.js に認証情報をロードする

Amazon EC2 インスタンスで Node.js アプリケーションを実行する場合、Amazon EC2 の IAM ロールを活用して自動的に認証情報をインスタンスに提供できます。IAM ロールを使用するようにイン

スタンスを設定する場合、SDK はアプリケーションの IAM 認証情報を自動的に選択するため、手動で認証情報を提供する必要がなくなります。

Amazon EC2 インスタンスへの IAM ロールの追加の詳細については、 IAM roles for Amazon EC2（Amazon EC2 の IAM ロール）を参照してください。

Node.js Lambda 関数の認証情報をロードする

AWS Lambda 関数を作成するときは、関数を実行するアクセス許可を持つ特別な IAM ロールを作成する必要があります。このロールは、実行ロールと呼ばれます。Lambda 関数を設定するときは、作成した IAM ロールを対応する実行ロールとして指定する必要があります。

実行ロールは、実行と他のウェブサービスを呼び出すために必要な認証情報を Lambda 関数に提供します。その結果、Lambda 関数内で記述した Node.js コードに認証情報を提供する必要はありません。

Lambda 実行ロール作成の詳細については、AWS Lambda Developer Guide（ デベロッパーガイド）のManage permissions: Using an IAM role (execution role)（許可の管理: IAM ロール (実行ロール)の使用）を参照してください。

ウェブブラウザで認証情報を設定する

ブラウザスクリプトから SDK に認証情報を提供する方法はいくつかあります。これらの中には、より安全なものもあれば、スクリプトの開発中により便利に使えるものもあります。

推薦順で認証情報を提供できる方法は次のとおりです。

1. Amazon Cognito アイデンティティを使用してユーザーを認証し、認証情報を提供する
2. ウェブフェデレーテッド ID を使用する

> **Warning**
>
> スクリプトで AWS 認証情報をハードコーディングすることはお勧めしません。認証情報をハードコーディングすると、アクセスキー ID とシークレットアクセスキーが公開される危険があります。

トピック

- Amazon Cognito ID を使用してユーザーを認証する

Amazon Cognito ID を使用してユーザーを認証する

ブラウザスクリプトの AWS 認証情報を取得するには、Amazon Cognito ID 認証情報クライアントを使用することをお勧めしますCognitoIdentityClient。Amazon Cognito では、サードパーティのアイデンティティプロバイダーによるユーザーの認証が可能です。

Amazon Cognito アイデンティティを使用するには、最初に Amazon Cognito コンソールでアイデンティティプールを作成する必要があります。ID プールは、アプリケーションがユーザーに提供する ID のグループを表します。ユーザーに与えられたアイデンティティは、各ユーザーアカウントを一意に識別します。Amazon Cognito ID は認証情報ではありません。これらは、 AWS Security Token Service () のウェブ ID フェデレーションサポートを使用して認証情報と交換されますAWS STS。

Amazon Cognito は、複数のアイデンティティプロバイダーにわたるアイデンティティの抽象化を管理するのに役立ちます。ロードされた ID は AWS STSの認証情報と交換されます。

Amazon Cognito ID 認証情報オブジェクトを設定する

まだ作成していない場合は、Amazon Cognito クライアントを設定する前に Amazon Cognito console (Amazon Cognito コンソール) でブラウザスクリプトによりアイデンティティプールを作成してください。アイデンティティプール用の認証済み IAM ロールと未認証 IAM ロールの両方を作成して関連付けます。詳細については、「Amazon Cognito デベロッパーガイド」の「チュートリアル: ID プールの作成」を参照してください。

認証されていないユーザーは アイデンティティが検証されないため、このロールはアプリケーションのゲストユーザーに適切です。または、ユーザーのアイデンティティが検証されているかどうかが重要ではない場合に適切です。認証されているユーザーは、自分の ID を確認するサードパーティーの ID プロバイダーを介してアプリケーションにログインします。リソースの許可の範囲を適切に設定し、認証されていないユーザーからのアクセスを許可しないようにします。

アイデンティティプールを設定したら、`@aws-sdk/credential-providers` から `fromCognitoIdentityPool` のメソッドを使用して、アイデンティティプールから認証情報を取得します。Amazon S3 クライアントを作成する次の例では、*`AWS_REGION`* をリージョンに、*`IDENTITY_POOL_ID`* をアイデンティティプール ID に置き換えます。

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

オプションの logins プロパティは、ID プロバイダー名の ID トークンへのマッピングです。ID プロバイダーからのトークンの取得方法は、使用するプロバイダーによって異なります。たとえば、Amazon Cognito ユーザープールを認証プロバイダーとして使用している場合は、以下のような方法を同様に使用できます。

```
// 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;
};
```

認証されていないユーザーを認証されたユーザーに切り替える

Amazon Cognito は、認証されたユーザーと認証されていないユーザーの両方をサポートします。認証されていないユーザーは、ID プロバイダーのいずれにもログインしていない場合でも、リソースにアクセスできます。このレベルのアクセスは、ログインする前にユーザーにコンテンツを表示するのに便利です。認証されていない各ユーザーは、個別にログインして認証していない場合でも Amazon Cognito で一意のアイデンティティを持ちます。

認証されていないユーザーとしての開始

ユーザーは通常、認証されていないロールから開始します。このロールでは、logins プロパティを使用しないで設定オブジェクトの認証情報プロパティを設定します。この場合、デフォルト認証情報は次のようになります。

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

認証されたユーザーへの切り替え

認証されていないユーザーが アイデンティティプロバイダーにログインしたときに、トークンがあれば、カスタム関数を呼び出して認証情報オブジェクトを更新し logins トークンを追加することで、認証されていないユーザーを認証されたユーザーに切り替えることができます。

```
// 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;
```

```
}
```

Node.js の考慮事項

Node.js コードは JavaScript ですが、Node.js AWS SDK for JavaScript での の使用は、ブラウザスクリプトでの SDK の使用とは異なる場合があります。一部の API メソッドは Node.js で動作しますが、ブラウザスクリプトでは動作しません。また、その逆も生じます。また、一部の API をうまく使用するには、File System (fs) モジュールなどの他の Node.js モジュールをインポートして使用するなど、一般的な Node.js コーディングパターンに精通している必要があります。

> **Note**
>
> AWS では、開発にアクティブ LTS バージョンの Node.js を使用することをお勧めします。

組み込み Node.js モジュールを使用する

Node.js では、インストールしなくても使用できる一連の組み込みモジュールを提供します。これらのモジュールを使用するには、モジュール名を指定するために require メソッドを使ってオブジェクトを作成します。たとえば、組み込みの HTTP モジュールを含めるには、次のようにします。

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

モジュールのメソッドを、そのオブジェクトのメソッドであるかのように呼び出します。たとえば、次に示すのは HTML ファイルを読み込むコードです。

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

Node.js が提供するすべての組み込みモジュールの完全なリストについては、Node.js ウェブサイトの「Node.js のドキュメント」を参照してください。

npm パッケージを使用する

組み込みのモジュールに加えて、Node.js パッケージマネージャーのnpmから、サードパーティーコードを含めたり、組み込んだりすることもできます。これは、オープンソースの Node.js パッケージとそれらのパッケージをインストールするためのコマンドラインインターフェイスのリポジトリです。npm の詳細と現在利用可能なパッケージのリストについては、https://www.npmjs.com を参照してください。 GitHub で使用可能な追加の Node.js パッケージについても確認できます。

Node.js で maxSockets を設定する

Node.js では、オリジンあたりの最大接続数を設定できます。 maxSockets が設定されている場合、低レベルの HTTP クライアントはリクエストをキューに入れ、利用可能になったときに、ソケットに割り当てます。

これにより、一度に特定のオリジンへの同時リクエスト数の上限を設定できます。この値を小さくすると、受信したスロットリングエラーまたはタイムアウトエラーの数を減らすことができます。ただし、ソケットが使用可能になるまでリクエストがキューに入れられるため、メモリ使用量も増加する可能性があります。

次の例は、DynamoDB クライアント用にmaxSocketsを設定する方法を示しています。

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

maxSockets 値またはAgentオブジェクトを指定しない場合、SDK for JavaScript は 50 の値を使用します。Agent オブジェクトを指定すると、そのmaxSockets値が使用されます。Node.js maxSocketsでの の設定の詳細については、Node.js ドキュメントを参照してください。

の v3.521.0 以降では AWS SDK for JavaScript、次の短縮構文を使用して を設定できますrequestHandler。

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

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

Node.js で keep-alive を使用して接続を再利用する

デフォルトの Node.js HTTP/HTTPS エージェントは新しいリクエストがあるたびに新しい TCP 接続を作成します。新しい接続を確立するコストを回避するために、 はデフォルトで TCP 接続を AWS SDK for JavaScript 再利用します。

Amazon DynamoDB クエリなどの短期間のオペレーションでは、TCP 接続を設定する際のレイテンシーのオーバーヘッドが、オペレーション自体よりも大きくなる可能性があります。さらに、保管時の DynamoDB 暗号化は と統合されているためAWS KMS、データベースからのレイテンシーがオペレーションごとに新しい AWS KMS キャッシュエントリを再確立しなければならない場合があります。 https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/encryption.howitworks.html

TCP 接続を再利用しない場合は、DynamoDB クライアントの次の例に示すように、サービスごとのクライアントベースkeepAliveで でこれらの接続をライブで再利用することを無効にできます。

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

keepAliveが有効になっている場合、TCPキープアライブパケットの初期遅延をkeepAliveMsecsに設定することもできます。これは、デフォルトでは1000ミリ秒(ms)です。詳細については、Node.js のドキュメントを参照してください。

Node.js のプロキシを設定する

インターネットに直接接続できない場合、SDK for JavaScript は サードパーティーの HTTP エージェントを介した HTTP または HTTPS プロキシの使用をサポートします。

サードパーティーの HTTP エージェントを検索するには、npmで「HTTP プロキシ」を検索します。

サードパーティーの HTTP エージェントプロキシをインストールするには、コマンドプロンプトで次のように入力します。ここでは*PROXY*（プロキシ）はnpmパッケージの名前です。

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

アプリケーションでプロキシを使用するには、DynamoDBクライアントの次の例に示すように、httpAgentと　httpsAgentプロパティを使用します。

```
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はhttpsAgentと同じではなく、クライアントからのほとんどの呼び出しは https になるため、両方を設定する必要があります。

Node.js に証明書バンドルを登録する

Node.js のデフォルトの信頼ストアには、 AWS のサービスへのアクセスに必要な証明書が含まれています 場合によっては、特定の証明書セットのみを含めることが望ましい場合があります。

この例では、指定された証明書が提供されない限り、接続を拒否する　https.Agent を作成するためにディスク上の特定の証明書が使用されます。新しく作成されたhttps.Agentは、その次に、DynamoDB クライアントによって使用されます。

```
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
  })
});
```

ブラウザスクリプトの考慮事項

以下のトピックでは、ブラウザスクリプト AWS SDK for JavaScript で を使用するための特別な考慮事項について説明します。

トピック

- SDK for Browsers を構築する
- クロスオリジンリソース共有 (CORS)
- アプリケーションを webpack にバンドルする

SDK for Browsers を構築する

SDK for JavaScriptバージョン2 (V2)とは異なり 、V3はサービスのデフォルトセットを含んでサポートする JavaScript ファイルとして提供されていません。代わりに V3 は、必要な SDK for JavaScript ファイルのみをバンドルしてブラウザに含め、オーバーヘッドの削減を有効にします。Webpack を使用して、必要な SDK for Java Script ファイルと、必要な追加のサードパーティパッケージを `Javascript`ファイルにまとめ、 `<script>`タグを使用してブラウザスクリプトにロードします。Webpackの詳細については、「アプリケーションを webpack にバンドルする」を参照してください。

ブラウザで CORS を強制する環境の外側で SDK を使用し、SDK for JavaScript によって提供されるすべてのサービスにアクセスする場合は、リポジトリを複製し、SDK のデフォルトのホストバー

ジョンをビルドするのと同じビルドツールを実行することで、SDK のカスタムコピーをローカルにビルドできます。次のセクションでは、追加のサービスと API バージョンを使用して SDK を構築するステップについて説明します。

SDK Builder を使用して SDK for JavaScript を構築する

> **Note**
>
> Amazon Web Services バージョン 3 (V3) はブラウザビルダーをサポートは終了しました。ブラウザアプリケーションの帯域幅の使用量を最小限に抑えるために、名前付きモジュールをインポートし、それらをバンドルしてサイズを小さくすることをお勧めします。バンドルの詳細については、「アプリケーションを webpack にバンドルする」を参照してください。

クロスオリジンリソース共有 (CORS)

Cross-Origin Resource Sharing (CORS) は、最新ウェブブラウザのセキュリティ機能です。これにより、ウェブブラウザはどのドメインが外部のウェブサイトまたはサービスのリクエストを行うことができるかをネゴシエートできます。

AWS SDK for JavaScript を使用してブラウザアプリケーションを開発する場合、CORS は重要な考慮事項です。リソースへのリクエストのほとんどは、ウェブサービスのエンドポイントなどの外部ドメインに送信されるためです。JavaScript 環境で CORS セキュリティが適用される場合は、そのサービスで CORS を設定する必要があります。

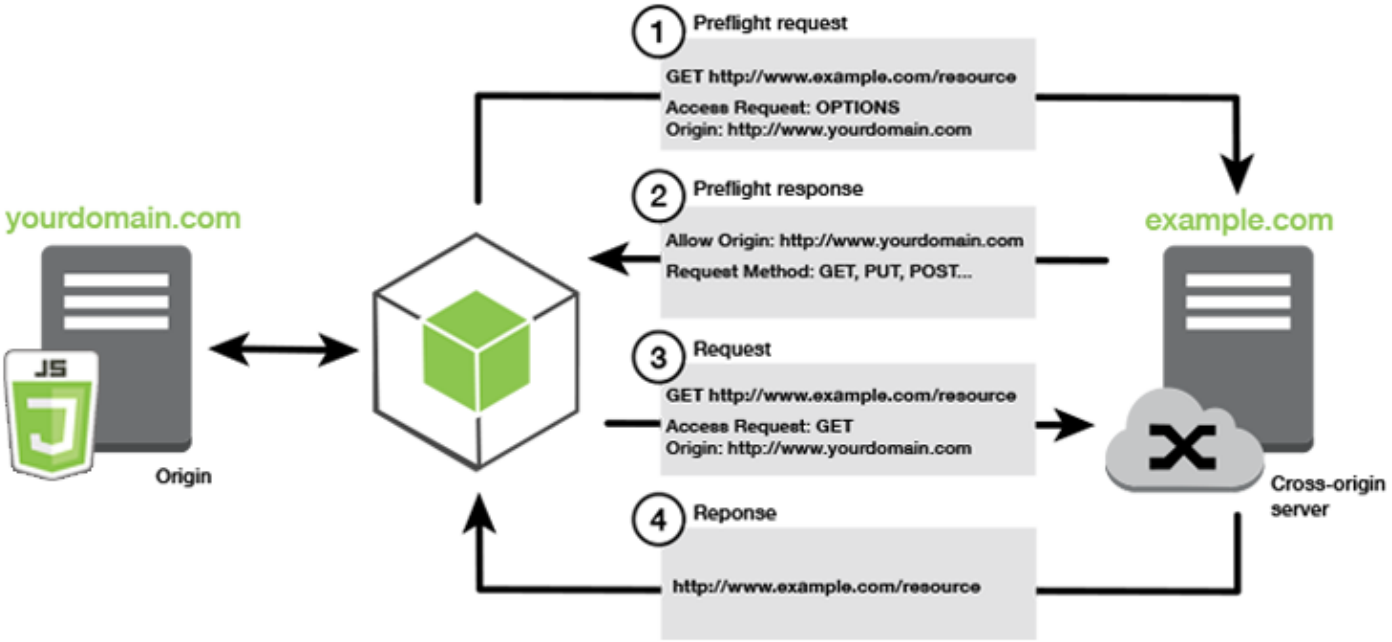
CORS は、クロスオリジンリクエストでリソースの共有を許可するかどうかを、以下に基づいて決定します。

- リクエストを行う特定のドメイン
- 行われている HTTP リクエストのタイプ (GET、PUT、POST、DELETE など)

CORS の仕組みは

最も簡単なケースとして、ブラウザスクリプトは他のドメインのサーバーからリソースの GET リクエストを行います。そのサーバーの CORS 設定に応じて、リクエストが GET リクエストの送信を許可されているドメインからのものである場合、クロスオリジンサーバーはリクエストされたリソースを返すことによって応答します。

リクエスト元のドメインまたは HTTP リクエストの種類のいずれかが承認されていない場合、リクエストは拒否されます。ただし、CORS では、実際に送信する前にリクエストをプリフライトすることができます。この場合、OPTIONS アクセスリクエストオペレーションが送信されるプリフライトリクエストが行われます。クロスオリジンサーバーの CORS 設定がリクエスト元ドメインへのアクセスを許可する場合、サーバーはリクエスト元ドメインがリクエストされたリソースに対して行うことができるすべての HTTP リクエストタイプをリストしたプリフライト応答を返送します。



CORS 設定は必要ですか？

Amazon S3 バケットを操作する前に、CORS の設定が必要です。一部のJavaScript環境では、CORSが適用されない場合があるため、CORSを設定する必要はありません。例えば、Amazon S3 バケットからアプリケーションをホストし、 `*.s3.amazonaws.com` またはその他の特定のエンドポイントからリソースにアクセスする場合、リクエストは外部ドメインにアクセスしません。したがって、この CORS 設定は必要ありません。この場合、CORS は Amazon S3 以外のサービスに使用されます。

Amazon S3 バケットの CORS を設定する

AmazonS3 コンソールで CORS を使用するように AmazonS3 バケットを設定できます。

AWS ウェブサービスマネジメントコンソールで CORS を設定する場合は、JSON を使用して CORS 設定を作成する必要があります。新しい AWS Web Services マネジメントコンソールは、JSON CORS 設定のみをサポートします。

> **Important**
>
> 新しい AWS Web Services マネジメントコンソールでは、CORS 設定は JSON である必要があります。

1. AWS ウェブサービスマネジメントコンソールで、Amazon S3 コンソールを開き、設定するバケットを見つけ、そのチェックボックスをオンにします。
2. 開いたペインで、Permissions (許可) を選択します。
3. Permissions (許可) タブで、CORS Configuration (CORS 設定) を選択します。
4. CORS Configuration Editor (CORS 設定エディタ) でCORS設定を入力して、Save (保存) を選択します。

CORS 設定は、<CORSRule> の一連のルールを含む XML ファイルです。設定は最大で 100 個のルールを持つことができます。ルールは次のいずれかのタグによって定義されます。

- <AllowedOrigin> –クロスドメインリクエストを許可するドメインオリジンを指定します。
- <AllowedMethod> –クロスドメインリクエストで許可するリクエストの種類 (GET、PUT、POST、DELETE、HEAD) を指定します。
- <AllowedHeader> –プリフライトリクエストで許可されるヘッダーを指定します。

設定例については、Amazon Simple Storage Service User Guide (Amazon Simple ストレージサービスユーザーガイド) の「 How do I configure CORS on my bucket? 」(バケットで CORS を設定する方法) を参照してください。

CORS 設定例

次の CORS 設定例では、ユーザーはドメインexample.orgからバケット内のオブジェクトを表示、追加、削除、または更新することができます。ただし、ウェブサイトのドメインに<AllowedOrigin>を追加することの検討をお勧めします。オリジンを許可するように "*" を指定できます。

> **Important**
>
> 新しい S3 コンソールでは、CORS 設定は JSON である必要があります。

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"]
    }
]
```

この設定では、ユーザーがバケットに対してアクションを実行することは許可されません。ブラウザのセキュリティモデルで Amazon S3 へのリクエストを許可できます。許可はバケット権限または IAM ロール権限を介して設定する必要があります。

ExposeHeader を使用して、SDK に Amazon S3 から返されたレスポンスヘッダーを読み込ませることができます。例えば、PUTまたはマルチパートアップロードからETagヘッダーを読み取る場合、前の例に示すように、設定にExposeHeaderタグを含める必要があります。SDK は、CORS 設定によって公開されているヘッダーにのみアクセスできます。オブジェクトにメタデータを設定すると、値は x-amz-meta-my-custom-header のように、プレフィックス x-amz-meta- を持つヘッダーとして返され、同じ方法で公開されている必要があります。

アプリケーションを webpack にバンドルする

ブラウザスクリプトまたはNode.jsでWebアプリケーションでコードモジュールを使用すると、依存関係が作成されます。これらのコードモジュールは独自の依存関係を持つことができ、結果として、アプリケーションが機能するために必要な、相互接続されたモジュールの集まりができます。依存関係を管理するには、webpackなどのモジュールバンドラーを使用できます。

webpackモジュールバンドラーはアプリケーションコードを解析して、import または require ステートメントを検索し、アプリケーションに必要なすべてのアセットを含むバンドルを作成します。これは、アセットを Web ページを介して簡単に提供できるようにするためです。SDK for JavaScript は、出力バンドルに含める依存関係の1つとしてwebpack に含めることができます。

webpackの詳細については、GitHub の webpack module bundler (Webpack モジュールバンドラー) を参照してください。

Webpackをインストールします

webpackモジュールバンドラーをインストールするには、まず npm (Node.js パッケージマネージャー) をインストールする必要があります。webpackCLI およびJavaScriptモジュールをインストールするには、次のコマンドを入力します。

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

ファイルとディレクトリパスを操作するためのモジュールpathを使用するには、Webpack で自動的にインストールされる場合は、Node.js path-browserifyパッケージをインストールする必要があります。

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

webpack を設定する

デフォルトで、webpack はプロジェクトのルートディレクトリにあるwebpack.config.jsという名前のJavaScript ファイルを検索します。このファイルは、設定オプションを指定します。WebPackバージョン 5.0.0とそれ 以降の webpack.config.js 設定ファイルの例を次に示します。

> **Note**
>
> Webpackの設定要件は、インストールする Webpack のバージョンによって異なります。詳細については、Webpack documentation (Webpack ドキュメント) を参照してください。

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

この例では、 entry point (エントリポイント) としてbrowser.js が指定されます。entry point (エントリポイント) は、インポートされたモジュールの検索を開始するために使用するファイ

ルwebpackです。出力のファイル名は bundle.js として指定されます。この出力ファイルには、アプリケーションの実行に必要なすべての JavaScript が含まれています。エントリーポイントで指定されたコードが SDK for JavaScript などの他のモジュールをインポートまたは必要とする場合、そのコードは設定で指定しなくてもバンドルされます。

webpack を実行する

アプリケーションがwebpackを使用するように設定するには、package.jsonファイル内のscriptsオブジェクトに以下を追加します。

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

以下は、webpackの追加を示すpackage.jsonファイルの例です。

```
{
  "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"
  }
}
```

アプリケーションを構築するには、次のコマンドを入力します。

```
npm run build
```

そして、webpackモジュールバンドラーは、プロジェクトのルートディレクトリで指定した JavaScript ファイルを生成します。

webpack バンドルを使用する

ブラウザスクリプトでバンドルを使用するには、次の例で示すように <script> タグを使用してバンドルを組み込むことができます。

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

Node.js のバンドル

設定でターゲットとしてnodeを指定することにより、Node.jsで実行されるバンドルを生成するためにwebpackを使用できます。

```
target: "node"
```

これは、ディスク容量に制限がある環境で Node.js アプリケーションを実行するときに役立ちます。出力ターゲットとして指定された Node.js を使用した webpack.config.js 設定の例を次に示します。

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

SDK for JavaScript での AWS サービスの操作

AWS SDK for JavaScript v3 は、クライアントクラスのコレクションを通じてサポートされるサービスへのアクセスを提供します。これらのクライアントクラスから、一般にサービスオブジェクトと呼ばれるサービスインターフェイスオブジェクトが作成されます。サポートされている各 AWS サービスには、サービスの機能とリソースを使用するための低レベル APIs を提供する 1 つ以上のクライアントクラスがあります。例えば、Amazon DynamoDB API は、DynamoDB クラスから利用できます。

SDK for JavaScript を通じて公開されるサービスは、リクエストレスポンスのパターンに従って、呼び出し元のアプリケーションとメッセージを交換します。このパターンでは、サービスを呼び出すコードが HTTP/HTTPS リクエストをそのサービスのエンドポイントに送信します。リクエストには、呼び出されている特定の機能を正常に呼び出すために必要なパラメータが含まれています。呼び出されたサービスは、リクエスタに返されるレスポンスを生成します。オペレーションが成功した場合、レスポンスにはデータが含まれています。オペレーションが失敗した場合は、エラー情報が含まれています。

AWS サービスの呼び出しには、試行された再試行を含む、サービスオブジェクトに対する オペレーションの完全なリクエストとレスポンスのライフサイクルが含まれます。リクエストには、JSON パラメータとして 0 個以上のプロパティが含まれています。レスポンスは、操作に関連するオブジェクトによってカプセル化され、コールバック機能や JavaScript promise などのいくつかの手法の1つを通してリクエスタに返信されます。

トピック

サービスオブジェクトを作成して呼び出す

JavaScript API は、利用可能なほとんどの AWS サービスをサポートしています。JavaScript APIの各サービスは、サービスがサポートするすべてのAPIを呼び出すために使用するsendメソッドをクライアントクラスに提供します。JavaScript API のサービスクラス、オペレーション、およびパラメータの詳細については、[API Reference]を参照してください。

Node.jsでSDKを使用する場合は、importを使用して、必要な各サービスのSDKパッケージをアプリケーションに追加します。これにより、現在のすべてのサービスがサポートされます。次の例では、us-west-1地域に Amazon S3 サービスオブジェクトを作成します。

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

サービスオブジェクトパラメータを指定する

サービスオブジェクトのメソッドを呼び出す場合、API の必要に応じて JSON でパラメータを渡します。例えば、Amazon S3 では、指定されたバケットとキーのオブジェクトを取得するには、 からGetObjectCommandメソッドに次のパラメータを渡しますS3Client。JSON パラメータを渡す詳細については、「JSON の使用」を参照してください。

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

Amazon S3 パラメータの詳細については、 API リファレンスの「@aws-sdk/client-s3」を参照してください。

TypeScript で生成されたクライアントに @smithy/types を使用する

TypeScript を使用している場合、 @smithy/typesパッケージを使用すると、クライアントの入出力シェイプを操作できます。

シナリオ: 入力構造と出力構造undefinedから を削除する

生成されたシェイプのメンバーは、入力シェイプundefinedの場合は と結合され、出力シェイプの場合は ? (オプション) です。入力の場合、検証はサービスに延期されます。出力の場合、出力データの実行時にチェックを行うことを強くお勧めします。

これらのステップをスキップする場合は、 AssertiveClientまたは UncheckedClient タイプのヘルパーを使用します。次の例では、Amazon S3 サービスでタイプヘルパーを使用します。

```
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();
```

Command 構文で非集約クライアントで変換を使用する場合、入力は、以下の例に示すように別のクラスを通過するため、検証できません。

```
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();
```

シナリオ: Smithy-TypeScript が生成したクライアントの出力ペイロード BLOB タイプを絞り込む

このシナリオは、 AWS SDK for JavaScript v3 の 内などS3Client、ストリーミング本文を使用するオペレーションに主に関連します。

BLOB ペイロードタイプはプラットフォームに依存するため、クライアントが特定の環境で実行されていることをアプリケーションに指定できます。これにより、次の例に示すように BLOB ペイロードタイプが絞り込まれます。

```
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!;
```

サービスを非同期的に呼び出す

SDK を介して行われるリクエストはすべて非同期です。ブラウザスクリプトを作成するときに、この点に注意してください。通常、ウェブブラウザで実行されている JavaScript には、1 つの実行スレッドしかありません。 AWS サービスに対して非同期呼び出しを行った後、ブラウザスクリプトは実行を継続し、プロセス内で、返される前にその非同期結果に依存するコードを実行しようとする可能性があります。

AWS サービスに対して非同期呼び出しを行うには、それらの呼び出しを管理し、データが使用可能になる前にコードがデータを使用しようとしないようにします。このセクションのトピックでは、非同期呼び出し管理の必要性を説明し、それらを管理するために使用できるさまざまな手法について詳しく説明します。

非同期呼び出しを管理するには、これらの方法のいずれかを使用できますが、すべての新しいコードに対して非同期/待機を使用することをお勧めします。

非同期/待機

V3 のデフォルトの動作であるため、この手法を使用することをお勧めします。

promise

非同期/待機 をサポートしないブラウザで、この手法を使用します。

コールバックします

非常に単純な場合を除き、コールバックの使用は避けてください。ただし、移行シナリオでは役立つ場合があります。

トピック

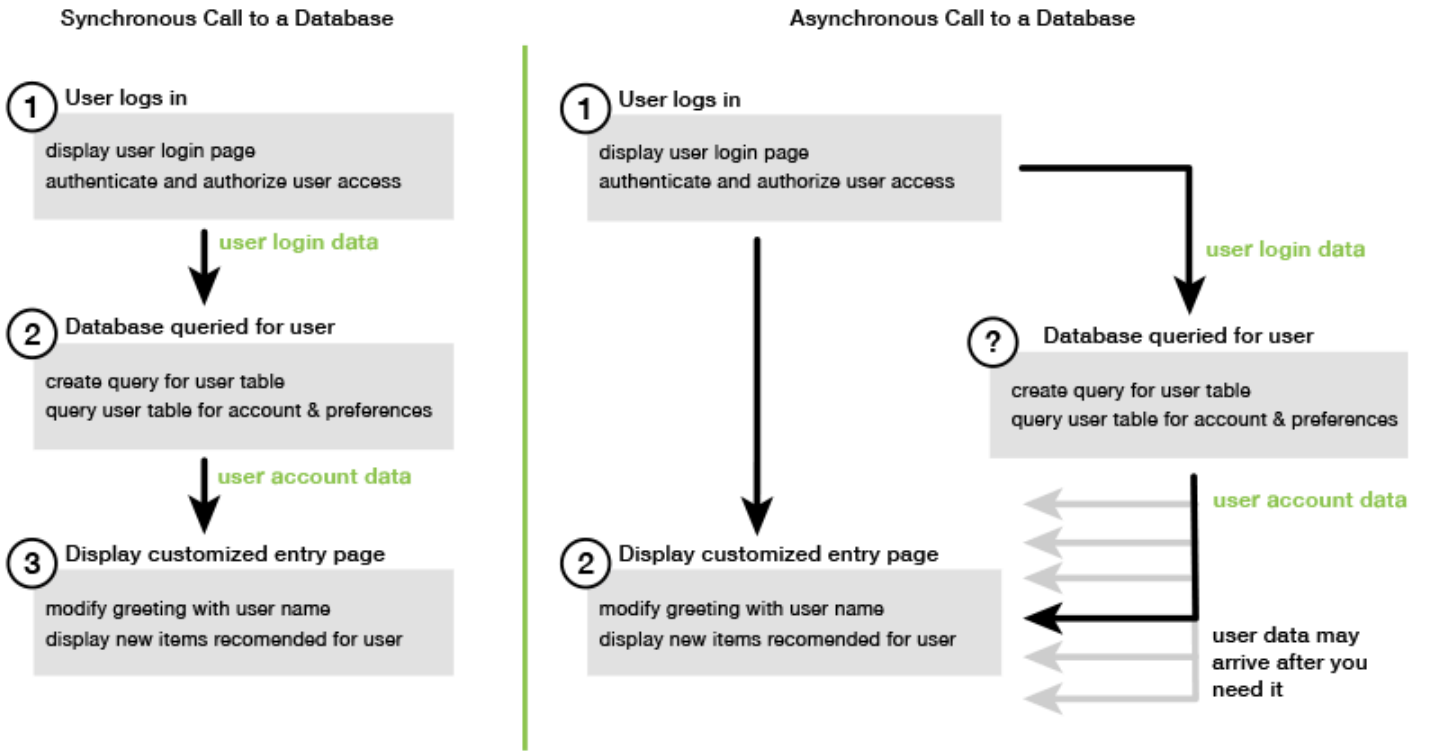
非同期呼び出しを管理する

たとえば、e コマースウェブサイトのホームページは、リピート顧客がサインインするようにします。サインインする顧客にとっての利点の 1 つは、サインイン後に、顧客の特定の好みに合わせてサイトがカスタマイズされることです。これを実現するには、以下のことが必要です。

1. 顧客はログインし、サインイン認証情報で認証を受ける必要があります。
2. 顧客の好みは顧客データベースからリクエストされます。
3. データベースは、ページがロードされる前に、サイトのカスタマイズに使用される顧客の好みを提供します。

これらのタスクが同期的に実行される場合、それぞれの処理が完了してからでなければ、次が開始できません。データベースから顧客選定が戻るまで、ウェブページはロードを終了することはできません。しかし、データベースクエリがサーバーに送信された後、ネットワークのボトルネック、異常に高いデータベーストラフィック、またはモバイルデバイスの接続不良のために、顧客データの受信が遅れたり、失敗することさえあります。

このような状況でウェブサイトがフリーズしないようにするため、データベースを非同期的に呼び出します。データベース呼び出しが実行され、非同期リクエストが送信された後も、コードは想定どおりに継続して実行されます。非同期呼び出しのレスポンスを適切に管理しないと、コードは、データベースから返されると想定される情報がまだ利用できないときに、そのデータを使用しようとする可能性があります。



async/await を使用する

promise を使用するのではなく、async/await の使用を検討する必要があります。async 関数は、promise を使用するよりもシンプルで、使用する定型コードが少なくなります。await は、async 関数内でのみ使用して、値を非同期的に待機することができます。

次の例では、async/await を使用して、すべての Amazon DynamoDB テーブルを `us-west-2` で一覧表示します。

> **Note**
>
> この例を実行するには、
>
> - プロジェクトのコマンドライン`npm install @aws-sdk/client-dynamodb`に を入力して、 AWS SDK for JavaScript DynamoDB クライアントをインストールします。
> - AWS 認証情報が正しく設定されていることを確認します。詳細については、「認証情報の設定」を参照してください。

```
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**
>
> すべてのブラウザが async/await をサポートしているわけではありません。非同期/待機をサポートするブラウザのリストについては、非同期関数を参照してください。

JavaScript promises を使用する

コールバックを使用する代わりに、サービスクライアントの AWS SDK for JavaScript v3 `ListTablesCommand` メソッド () を使用してサービス呼び出しを行い、非同期フローを管理します。次の例は、「Amazon DynamoDB」(Amazon DynamoDB)表の名前を`us-west-2`で取得する方法を示しています。

```
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);
  });
```

複数の promise を調整する

状況によって、コードは複数の非同期呼び出しを行う必要があります。すべてが正常に返されたときのみ、これらの呼び出しに対する操作が必要です。これらの個々の非同期メソッド呼び出しを promises で管理する場合、`all` メソッドを使用する追加の promise を作成することができます。

このメソッドは、ユーザーがメソッドに渡す promise の配列が満たされた場合に、この包括的な promise を満たします。コールバック関数には、`all` メソッドに渡された promises の値の配列が渡されます。

次の例では、 AWS Lambda 関数は Amazon DynamoDB に対して 3 つの非同期呼び出しを行う必要がありますが、各呼び出しの promise が満たされた後にのみ完了できます。

```
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;
```

ブラウザおよび Node.js による Promises のサポート

ネイティブ JavaScript の promises (ECMAScript 2015) のサポートは、コードが実行される JavaScript エンジンとバージョンによって異なります。コードを実行する必要がある各環境における JavaScript Promise のサポートを確認するには、GitHub の「ECMAScript 互換性の表」を参照してください。

匿名コールバック関数を使用する

各サービスオブジェクトメソッドは、最後のパラメータとして無名コールバック機能を受け取ることができます。このコールバック機能の特性は次のとおりです。

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

このコールバック関数が実行されるのは、成功したレスポンスまたはエラーデータが返されたときです。メソッドの呼び出しに成功すると、レスポンスの内容は `data` パラメータでコールバック関数に

利用可能になります。呼び出しが成功しない場合、エラーの詳細は `error` パラメータに記載されます。

通常、コールバック関数内のコードはエラーをテストし、エラーが返された場合はそれを処理します。エラーが返されない場合、コードは `data` パラメータからレスポンス内のデータを取得します。コールバック関数の基本的な形式は次の例のようになります。

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

前の例では、エラーまたは返されたデータの詳細がコンソールのログに記録されます。サービスオブジェクトのメソッド呼び出しの一部として渡されるコールバック関数の例を、次に示します。

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

サービスクライアントリクエストを作成する

AWS サービスクライアントへのリクエストは簡単です。SDK for JavaScripのバージョン 3 (V3) では、リクエストの送信を行うことができます。

> **Note**
>
> SDK for JavaScript の V3 を使用する場合は、バージョン 2 (V2)のコマンドを使用して操作を行うこともできます。詳細については、「v2 コマンドの使用」を参照してください。

リクエストを送信するには

1. クライアントオブジェクトを特定の AWS 地域など、希望する設定で初期化します。
2. (オプション) 特定の Amazon S3 バケットの名前など、リクエストの値を持つリクエストJSONオブジェクトを作成します。クライアントメソッドに関連付けられた名前を持つインターフェイスの API リファレンスのトピックを参照して、リクエストのパラメータを調べることができます。たとえば、*abcCommand*クライアントクライアントメソッドを使用した場合、リクエストインターフェイスは *ABCInput*となります。
3. オプションで、リクエストオブジェクトを入力としてサービスコマンドを初期化します。
4. コマンドオブジェクトを入力として、クライアントでsendを呼び出します。

たとえば、Amazon DynamoDB テーブルをus-west-2で一覧表示するには、非同期/待機で実行できます。

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

サービスクライアントのレスポンスを処理する

サービスクライアントメソッドが呼び出されると、クライアントメソッドに関連付けられた名前を持つインタフェースのレスポンスオブジェクトインスタンスを返信します。例えば、*AbcCommand* クライアントメソッドを使用した場合、レスポンスオブジェクトは *AbcResponse* (インタフェース) タイプになります。

レスポンスで返されたデータにアクセスする

レスポンスオブジェクトには、サービスリクエストによって戻されたデータがプロパティとして含まれています。

サービスクライアントリクエストを作成するでは、 `ListTablesCommand`コマンドがレスポンスの`TableNames`のプロパティのテーブル名を返しました。

アクセスエラー情報

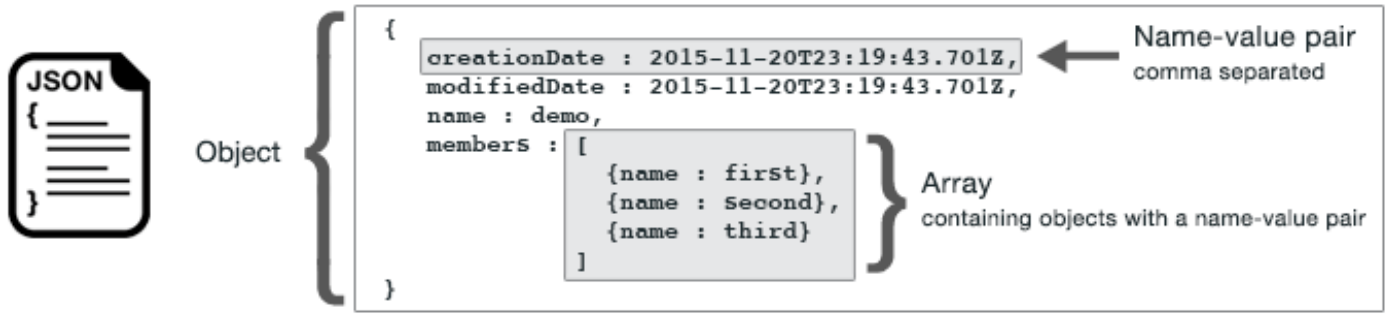
コマンドが失敗した場合、例外が発生します。次のコードスニペットは、サービス例外を処理する方法を示しています。

```
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
  }
}
```

JSON の使用

JSON は、人間にも機械にも読み取り可能な、データ交換のためのフォーマットです。JSON という名前は、[JavaScript Object Notation]の頭字語ですが、JSON の形式はどのプログラミング言語にも依存しません。

は JSON AWS SDK for JavaScript を使用して、リクエストを行うときにサービスオブジェクトにデータを送信し、サービスオブジェクトからデータを JSON として受信します。JSON の詳細については、json.org を参照してください。



JSON は 2 つの方法でデータを表します。

- [object]としては、名前と値のペアの順不同のコレクションです。オブジェクトは左中括弧 ({) と右中括弧 (}) で囲んで定義します。それぞれの名前と値のペアは名前で始まり、続けてコロン、その後に値が続きます。名前と値のペアはカンマで区切ります。
- [array]としては、値の順序付けられたコレクションです。配列は左角括弧 ([) と右角括弧 (]) で囲んで定義します。配列の項目はカンマで区切ります。

これは、オブジェクトの配列を含む JSON オブジェクトの例です。オブジェクトは、カードゲームのカードを表しています。各カードは 2 つの名前と値のペアで定義されます。1 つはそのカードを識別するための一意の値を指定し、もう 1 つは対応するカードイメージを指す URL を指定します。

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

サービスオブジェクトパラメータとしての JSON

これは、 AWS Lambda サービスオブジェクトへの呼び出しのパラメーターを定義するために使用される単純なJSONの例です。

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

params オブジェクトは、左右の中括弧内にコンマで区切られた、3 つの名前と値のペアによって定義されています。サービスオブジェクトメソッドの呼び出しにパラメータを指定する場合、呼び出す予定のサービスオブジェクトメソッドのパラメータ名によって名前が決まります。Lambda 関数を呼び出すとき、FunctionName、Payload、および LogType は、Lambda サービスオブジェクトの invoke メソッドの呼び出しに使用されるパラメータです。

サービスオブジェクトのメソッド呼び出しにパラメータを渡すとき、次の Lambda 関数の呼び出しの例に示されているように、メソッド呼び出しに JSON オブジェクトを渡します。

```
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 };
};
```

AWS SDK for JavaScript 通話のログ記録

AWS SDK for JavaScript には組み込みロガーが組み込まれているため、SDK for JavaScript で行った API コールをログに記録できます。

ロガーを有効にしてコンソールでログエントリを印刷するには、オプションの loggerパラメータを使用してサービスクライアントを設定します。次の例では、トレース出力とデバッグ出力を無視しながら、クライアントログ記録を有効にします。

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

ミドルウェアを使用したリクエストのログ記録

AWS SDK for JavaScript はミドルウェアスタックを使用して、オペレーション呼び出しのライフサイクルを制御します。スタック内の各ミドルウェアは、リクエストオブジェクトに変更を加えた後に次のミドルウェアを呼び出します。また、どのミドルウェアが呼び出されてエラーが発生したかを正確に把握できるため、スタック内の問題のデバッグもはるかに簡単になります。ミドルウェアを使用したリクエストのログ記録の例を次に示します。

```
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({});
```

上記の例では、ミドルウェアが DynamoDB クライアントのミドルウェアスタックに追加されています。最初の引数は、 を受け入れる関数`next`、呼び出すスタック内の次のミドルウェア、および呼び出されるオペレーションに関するいくつかの情報を含む`context`オブジェクトです。これは、 を受け入れる関数`args`、 オペレーションと リクエストに渡されたパラメータを含むオブジェクト、および で次のミドルウェアを呼び出す結果を返します`args`。

DynamoDB で AWS アカウントベースのエンドポイントを使用する

DynamoDB はAWS 、アカウント ID を使用してリクエストのルーティングを合理化することで、パフォーマンスを向上させることができるアカウントベースのエンドポイントを提供します。 AWS

この機能を利用するには、バージョン 3.656.0 以降の AWS SDK for JavaScript バージョン 3 を使用する必要があります。このアカウントベースのエンドポイント機能は、この新しいバージョンではデフォルトで有効になっています。

アカウントベースのルーティングをオプトアウトする場合は、次のオプションがあります。

- `accountIdEndpointMode` パラメータを に設定して、DynamoDB サービスクライアントを設定しますdisabled。
- 環境変数を `AWS_ACCOUNT_ID_ENDPOINT_MODE`に設定しますdisabled。
- 共有 AWS 設定ファイル設定を `account_id_endpoint_mode`に更新しますdisabled。

次のスニペットは、DynamoDB サービスクライアントを設定してアカウントベースのルーティングを無効にする方法の例です。

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

AWS SDKsおよびツールリファレンスガイドには、他の設定オプションに関する詳細が記載されています。

Amazon S3 チェックサムによるデータ整合性保護

Amazon Simple Storage Service (Amazon S3) では、オブジェクトをアップロードするときにチェックサムを指定できます。チェックサムを指定すると、そのチェックサムはオブジェクトとともに保存され、オブジェクトのダウンロード時に検証できます。

チェックサムは、ファイルを転送する際のデータの整合性をさらに強化します。チェックサムを使用すると、受信したファイルが元のファイルと一致することを確認することで、データ整合性を検証できます。Amazon S3 でのチェックサムの詳細については、サポートされているアルゴリズムを含む Amazon Simple Storage Service ユーザーガイドを参照してください。

ニーズに最適なアルゴリズムを柔軟に選択して、SDK にチェックサムを計算させることができます。または、サポートされているアルゴリズムのいずれかを使用して、事前に計算されたチェックサム値を指定することもできます。

> **Note**
>
> のバージョン 3.729.0 以降 AWS SDK for JavaScript、SDK はアップロードのCRC32チェックサムを自動的に計算することで、デフォルトの整合性保護を提供します。事前計算されたチェックサム値を指定しない場合、または SDK がチェックサムの計算に使用するアルゴリズムを指定しない場合、SDK はこのチェックサムを計算します。
> SDK には、外部で設定できるデータ整合性保護のグローバル設定も用意されています。詳細については、 AWS SDKsおよびツールリファレンスガイドを参照してください。

オブジェクトのアップロード

の PutObject コマンドを使用してAmazon S3 にオブジェクトをアップロードしますS3Client。のビルダーの ChecksumAlgorithmパラメータを使用してPutObjectRequest、チェックサム計算を有効にし、アルゴリズムを指定します。有効な値については、サポートされているチェックサムアルゴリズムを参照してください。

次のコードスニペットは、CRC-32 チェックサムを含むオブジェクトをアップロードするリクエストを示しています。SDK はリクエストを送信すると、CRC-32 チェックサムを計算してオブジェクトをアップロードします。Amazon S3 はオブジェクトと共にチェックサムを保存します。

```
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,
});
```

リクエストでチェックサムアルゴリズムを指定しない場合、チェックサムの動作は、次の表に示すように、使用する SDK のバージョンによって異なります。

チェックサムアルゴリズムが指定されていない場合のチェックサムの動作

SDK for JavaScript バージョン	チェックサムの動作
3.729.0 より前	SDK は CRC ベースのチェックサムを自動的に計算せず、リクエストで指定します。
3.729.0 以降	SDK はCRC32アルゴリズムを使用してチェックサムを計算し、リクエストで提供します。Amazon S3 は、独自のCRC32チェックサムを計算して転送の整合性を検証し、SDK が提供するチェックサムと比較します。チェックサムが一致した場合、チェックサムは オブジェクトとともに保存されます。

SDK が計算するチェックサムが、Amazon S3 がリクエストを受信したときに計算するチェックサムと一致しない場合、エラーが返されます。

事前に計算されたチェックサム値を使用してください。

リクエストで事前に計算されたチェックサム値を指定すると、SDK による自動計算が無効になり、代わりに提供された値が使用されます。

次の例は、事前に計算された SHA-256 チェックサムを含むリクエストを示しています。

```
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,
});
```

Amazon S3 が、指定されたアルゴリズムのチェックサム値が正しくないと判断した場合、サービスはエラーレスポンスを返します。

マルチパートアップロード

チェックサムはマルチパートアップロードでも使用できます。 AWS SDK for JavaScript は、 からへのUploadライブラリオプションを使用して@aws-sdk/lib-storage、マルチパートアップロードでチェックサムを使用できます。

```
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 for JavaScript のコードサンプル

このセクションのトピックには、さまざまな サービスの APIs AWS SDK for JavaScript で を使用して一般的なタスクを実行する方法の例が含まれています。

これらの例などのソースコードについては、[AWS [Code Examples Repository on GitHub]で見つけてください。 AWS ドキュメントチームが作成を検討するための新しいコード例を提案するには、リクエストを作成します。チームは、個々の API 呼び出しのみを対象とするシンプルなコードスニペットよりは、より広範なシナリオやユースケースを対象とするコード例を作成しようとしています。手順については、「GitHub の投稿ガイドライン」の「コードの作成」セクションを参照してください。

> **Important**
>
> これらの例では、ECMAScript6 のimport/export 構文を使用します。
>
> - これには Node.js バージョン 14.17 以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、Node.js ダウンロードを参照してください。
> - CommonJS 構文を使用する場合はJavaScript ES6/CommonJS 構文 for conversion guidelinesを参照してください。

トピック

JavaScript ES6/CommonJS 構文

AWS SDK for JavaScript コード例は ECMAScript 6 (ES6) で記述されています。ES6 は、コードをよりモダンで読みやすくし、より多くのことをおこなうための新しい構文と新機能を提供します。

ES6 では Node.js バージョン 13.x 以降を使用する必要があります。Node.js の最新バージョンをダウンロードしてインストールするには、Node.js downloads. を参照してください。ただし、必要に応じて、次のガイドラインを使用して CommonJS 構文に例を変換することができます。

- プロジェクト環境の `package.json` から `"type" : "module"` を削除します。

- すべての ES6 `import` ステートメントを CommonJS `require` ステートメントに変換してください。例えば、変換します、

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

CommonJS に相当する:

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

- すべての ES6 `export` ステートメントを CommonJS `module.exports` ステートメントに変換してください。例えば、変換します、

```
export {s3}
```

CommonJS に相当する:

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

次の例は、ES6 と CommonJS の両方で Amazon S3 バケットを作成するためのコード例を示しています。

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 例

AWS Elemental MediaConvert は、ブロードキャストグレード機能を備えたファイルベースのビデオトランスコーディングサービスです。このサービスでは、インターネット全体に配信するブロードキャストおよびビデオオンデマンド (VOD) 用のアセットを作成できます。詳細については、AWS Elemental MediaConvert ユーザーガイドをご参照ください。

JavaScript API for MediaConvert は `MediaConvert` クライアントクラスを通じて公開されます。詳細については、API リファレンスの「Class: MediaConvert」(クラス: MediaConvert)を参照してください。

トピック

- MediaConvert でのコード変換ジョブの作成と管理
- MediaConvert でジョブテンプレートを使用

MediaConvert でのコード変換ジョブの作成と管理

この Node.js コード例は以下を示しています。

- MediaConvert で使用するリージョン固有のエンドポイントを指定する方法。
- MediaConvert でコード変換ジョブを作成する方法。
- コード変換ジョブをキャンセルする方法。
- 完了したコード変換ジョブの JSON を取得する方法。
- 最近作成されたジョブの最大 20 個の JSON 配列を取得する方法。

シナリオ

この例では、Node.js モジュールを使用して MediaConvert を呼び出し、コード変換ジョブを作成および管理します。コードは SDK for JavaScript を使用して、MediaConvert クライアントクラスのこれらのメソッドを使用してこれを取得します。

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

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了します。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

- ジョブの入力ファイル用および出力ファイル用のストレージを提供する Amazon S3 バケットを作成して設定します。詳細については、「AWS Elemental MediaConvert ユーザーガイド」の「ファイルのストレージを作成する」を参照してください。
- 入力動画を、入力ストレージ用にプロビジョニングした Amazon S3 バケットにアップロードします。サポートされている入力動画のコーデックとコンテナの一覧については、「AWS Elemental MediaConvert ユーザーガイド」の「サポートされる入力コーデックおよびコンテナ」を参照してください。
- MediaConvert に入力ファイルと、出力ファイルが保存されている Amazon S3 バケットへのアクセスを付与する IAM ロールを作成します。詳細については、「AWS Elemental MediaConvert ユーザーガイド」の「IAM アクセス許可の設定」を参照してください。

> **Important**
>
> この例では、ECMAScript6 (ES6) を使用しています。これには Node.js バージョン13.x 以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> ただし、CommonJS 構文を使用したい場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

SDK の設定

前述のように、必要なクライアントとパッケージのダウンロードを含め、SDKを設定します。MediaConvert は、アカウントごとにカスタムエンドポイントを使用します。したがって、リージョン固有のエンドポイントを使用するために `MediaConvert` クライアントクラスも設定する必要があります。これを行うには、`mediaconvert(endpoint)` で `endpoint` パラメータを設定します。

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

シンプルなコード変換ジョブの定義

`libs`ディレクトリを作成し、ファイル名`emcClient.js`でNode.js モジュールを作成します。それに以下のコードをコピーし、ペーストしてMediaConvert クライアントオブジェクトを作成します。*`REGION`* を自分の AWS リージョンに置き換えます。*`ENDPOINT`*をMediaConvertアカウントエ

ンドポイントに置き換えます。これは、MediaConvertコンソールのアカウントページで確認できます。

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

このサンプルコードは、このGitHubにで見つけられます。

emc_createjob.jsというファイル名で Node.js モジュールを作成します。前に示したように、必要なクライアントとパッケージをインストールに含め、SDK が設定されていることを確認します。コード変換ジョブのパラメータを定義する JSON を作成します。

これらは非常に詳細なパラメータです。AWS Elemental MediaConvert コンソールを使用して JSON ジョブのパラメータを生成できます。そのためには、コンソールでジョブ設定を選択し、[ジョブ] セクションの下部にある [ジョブ JSON の表示] を選択します。次の例は、シンプルなジョブの JSON を示しています。

> **Note**
>
> *JOB_QUEUE_ARN* をMediaConvert ジョブキューに、*IAM_ROLE_ARN*をIAM ロールの Amazon リソースネーム (ARN)に、*OUTPUT_BUCKET_NAME*を宛先バケット名-たとえば、「s3: //OUTPUT_BUCKET_NAME/」、および*[INPUT_BUCKET_AND_FILENAME]* を入力バケットとファイル名をたとえば、「s3: //INPUT_BUCKET/FILE_NAME」に置き換えます。

```
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",
    },
  },
};
```

コード変換ジョブの作成

ジョブパラメータJSONを作成した後、非同期runメソッドを呼び出してMediaConvertクライアントサービスオブジェクトを呼び出し、パラメータを渡します。作成されたジョブの ID がレスポンスの data で返されます。

```
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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node emc_createjob.js
```

このサンプルコードは、このGitHubにで見つけられます。

コード変換ジョブのキャンセル

libsディレクトリを作成し、ファイル名emcClient.jsでNode.js モジュールを作成します。それに以下のコードをコピーし、ペーストしてMediaConvert クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。*ENDPOINT*をMediaConvertアカウントエンドポイントに置き換えます。これは、MediaConvertコンソールのアカウントページで確認できます。

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

このサンプルコードは、このGitHubにで見つけられます。

emc_canceljob.jsというファイル名で Node.js モジュールを作成します。前述のように、必要なクライアントとパッケージのダウンロードに含めて、SDK が設定されていることを確認します。キャンセルするジョブの ID を含む JSON を作成します。次に、MediaConvertクライアントサービスオブジェクトを呼び出すための promise を作成してCancelJobCommandメソッドを呼び出し、パラメータを渡します。promise コールバックのレスポンスを処理します。

> **Note**
>
> *JOB_ID*をキャンセルするジョブのID に置き換えます。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node ec2_canceljob.js
```

このサンプルコードは、このGitHubにで見つけられます。

最新のコード変換ジョブの一覧表示

libsディレクトリを作成し、ファイル名emcClient.jsでNode.js モジュールを作成します。それに以下のコードをコピーし、ペーストしてMediaConvert クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。*ENDPOINT*をMediaConvertアカウントエンドポイントに置き換えます。これは、MediaConvertコンソールのアカウントページで確認できます。

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

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

このサンプルコードは、このGitHubにで見つけられます。

emc_listjobs.jsというファイル名で Node.js モジュールを作成します。前に示したように、必要なクライアントとパッケージをインストールに含め、SDK が設定されていることを確認します。

リストをASCENDINGまたはDESCENDINGの順序で並べ替えるかどうかを指定する値、チェックするジョブキューのAmazonリソース名（ARN）、および含めるジョブのステータスを含むパラメーターJSONを作成します。次に、MediaConvertクライアントサービスオブジェクトを呼び出すための promise を作成してListJobsCommandメソッドを呼び出し、パラメータを渡します。

> **Note**
>
> *QUEUE_ARN* をチェックするジョブキューの Amazon リソースネーム (ARN) に、*[STATUS]*（状態）をキューのステータスに置き換えます。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node emc_listjobs.js
```

このサンプルコードは、このGitHubにで見つけられます。

MediaConvert でジョブテンプレートを使用

この Node.js コード例は以下を示しています。

- AWS Elemental MediaConvert ジョブテンプレートを作成する方法。
- コード変換ジョブを作成するためのジョブテンプレートを使用する方法。
- すべてのジョブテンプレートを一覧表示する方法。
- ジョブテンプレートを作成する方法。

シナリオ

MediaConvert でコード変換ジョブを作成するために必要な JSON は詳細で、多数の設定が含まれています。後続のジョブを作成するために使用できるジョブテンプレートに既知の正常な設定を保存することで、ジョブ作成を大幅に簡素化できます。この例では、Node.js モジュールを使用して MediaConvert を呼び出し、ジョブテンプレートを作成、使用、および管理します。コードは SDK for JavaScript を使用して、MediaConvert クライアントクラスのこれらのメソッドを使用してこれを実行します。

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

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了します。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。
- MediaConvert に入力ファイルと、出力ファイルが保存されている Amazon S3 バケットへのアクセスを付与する IAM ロールを作成します。詳細については、「AWS Elemental MediaConvert ユーザーガイド」の「IAM アクセス許可の設定」を参照してください。

> **Important**
>
> これらの例では ECMAScript6 (ES6) を使用しています。これには Node.js バージョン 13.x 以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> ただし、CommonJS 構文を使用したい場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

ジョブテンプレートの作成

libsディレクトリを作成し、ファイル名emcClient.jsでNode.js モジュールを作成します。それに以下のコードをコピーし、ペーストしてMediaConvert クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。*ENDPOINT*をMediaConvertアカウントエンドポイントに置き換えます。これは、MediaConvertコンソールのアカウントページで確認できます。

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

```
export { emcClient };
```

このサンプルコードは、このGitHubにで見つけられます。

emc_create_jobtemplate.jsというファイル名で Node.js モジュールを作成します。前に示したように、必要なクライアントとパッケージをインストールに含め、SDK が設定されていることを確認します。

テンプレート作成用の JSON パラメータを指定します。以前の成功したジョブの JSON パラメータの大部分を使用して、テンプレートの Settings 値を指定できます。この例では、MediaConvert でのコード変換ジョブの作成と管理 のジョブ設定を使用します。

次に、MediaConvertクライアントサービスオブジェクトを呼び出すための promise を作成してCreateJobTemplateCommandメソッドを呼び出し、パラメータを渡します。

> **Note**
>
> *JOB_QUEUE_ARN* をチェックするジョブキューの Amazon リソースネーム (ARN) に、*BUCKET_NAME* を宛先 Amazon S3 バケットの名前に置き換えます。たとえば、「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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node emc_create_jobtemplate.js
```

このサンプルコードは、このGitHubに で見つけられます。

ジョブテンプレートからコード変換ジョブを作成します

libsディレクトリを作成し、ファイル名emcClient.jsでNode.js モジュールを作成します。それに以下のコードをコピーし、ペーストしてMediaConvert クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。*ENDPOINT*をMediaConvertアカウントエンドポイントに置き換えます。これは、MediaConvertコンソールのアカウントページで確認できます。

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

このサンプルコードは、このGitHubにで見つけられます。

emc_template_createjob.jsというファイル名で Node.js モジュールを作成します。前に示したように、必要なクライアントとパッケージをインストールに含め、SDK が設定されていることを確認します。

使用するジョブテンプレートの名前、使用する Settings など、作成するジョブに固有のジョブ作成パラメータ JSON を作成します。次に、CreateJobsCommandクライアントサービスオブジェクトを呼び出すための promise を作成してMediaConvertメソッドを呼び出し、パラメータを渡します。

> **Note**
>
> *JOB_QUEUE_ARN*をチェックするジョブキューの Amazon リソースネーム (ARN) に、*KEY_PAIR_NAME*を、*TEMPLATE_NAME*を、*ROLE_ARN*をロールのAmazonリソース名（ARN）に、そして*INPUT_BUCKET_AND_FILENAME*を入力バケットとファイル名に - たとえば、「s3：// BUCKET_NAME / FILE_NAME」、に置き換えます。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node emc_template_createjob.js
```

このサンプルコードは、このGitHubにで見つけられます。

ジョブテンプレートの一覧表化

libsディレクトリを作成し、ファイル名emcClient.jsでNode.js モジュールを作成します。それに以下のコードをコピーし、ペーストしてMediaConvert クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。*ENDPOINT*をMediaConvertアカウントエンドポイントに置き換えます。これは、MediaConvertコンソールのアカウントページで確認できます。

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

このサンプルコードは、このGitHubにで見つけられます。

emc_listtemplates.jsというファイル名で Node.js モジュールを作成します。前に示したように、必要なクライアントとパッケージをインストールに含め、SDK が設定されていることを確認します。

MediaConvert クライアントクラスの listTemplates メソッドで空のリクエストパラメータを渡すためのオブジェクトを作成します。一覧表示するテンプレート (NAME、CREATION DATE、SYSTEM)、一覧表示するテンプレートの数、およびそれらのソート順を決定するための値を含めます。ListTemplatesCommand メソッドを呼び出すには、MediaConvert クライアントサービスオブジェクトを呼び出すための promise を作成し、パラメータを渡します。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node emc_listtemplates.js
```

このサンプルコードは、このGitHubにで見つけられます。

ジョブテンプレートの削除

libsディレクトリを作成し、ファイル名emcClient.jsでNode.js モジュールを作成します。それに以下のコードをコピーし、ペーストしてMediaConvert クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。*ENDPOINT*をMediaConvertアカウントエンドポイントに置き換えます。これは、MediaConvertコンソールのアカウントページで確認できます。

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

このサンプルコードは、このGitHubにで見つけられます。

emc_deletetemplate.jsというファイル名で Node.js モジュールを作成します。前に示したように、必要なクライアントとパッケージをインストールに含め、SDK が設定されていることを確認します。

削除するジョブテンプレートの名前を MediaConvert クライアントクラスの DeleteJobTemplateCommand メソッドのパラメータとして渡すオブジェクトを作成します。DeleteJobTemplateCommand メソッドを呼び出すには、MediaConvert クライアントサービスオブジェクトを呼び出すための promise を作成し、パラメータを渡します。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node emc_deletetemplate.js
```

このサンプルコードは、このGitHubにで見つけられます。

AWS Lambda 例

AWS Lambda は、サーバーのプロビジョニングや管理、ワークロード対応のクラススケーリングロジックの作成、イベント統合の維持、ランタイムの管理を行わずにコードを実行できるようにするサーバーレスコンピューティングサービスです。

の JavaScript API AWS Lambda は、LambdaService クライアントクラスを介して公開されます。

v3 で Lambda AWS SDK for JavaScript 関数を作成して使用する方法を示す例のリストを次に示します。

- API Gateway を使用した Lambdaを呼び出し
- AWS Lambda 関数を実行するためのスケジュールされたイベントの作成

Amazon Lex での例

Amazon Lex は、音声とテキストを使用して会話型インターフェイスをアプリケーションに組み込むための AWS サービスです。

JavaScript API for Amazon Lex はLex Runtime Service（Lexランタイムサービス）クライアントクラスを介して公開されます。

- Amazon Lex chatbotを構築する

Amazon Pollyの例

この Node.js コード例は以下を示しています。

- Amazon Polly を使用して録音した音声をAmazon S3 にアップロードします

シナリオ

この例では、一連のNode.jsモジュールを使用して、AmazonS3クライアントクラスのこれらのメソッドを使用して「Amazon Polly」(Amazon Polly)を使用して録音されたオーディオをAmazonS3に自動的にアップロードします。

- `StartSpeechSynthesisTaskCommand`

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- GitHubの手順に従って、Node JavaScriptの例を実行するようにプロジェクト環境を設定します。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。
- AWS Identity and Access Management (IAM) 認証されていない Amazon Cognito ユーザーロール polly:SynthesizeSpeech アクセス許可と、IAM ロールがアタッチされた Amazon Cognito ID プールを作成します。を使用して AWS リソースを作成する AWS CloudFormationのセクションでは、これらのリソースを作成する方法について以下のことを説明します。

> **Note**
>
> この例では Amazon Cognito を使用していますが、Amazon Cognito を使用していない場合、 AWS ユーザーは次の IAM アクセス許可ポリシーを持っている必要があります
>
> JSON

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

を使用して AWS リソースを作成する AWS CloudFormation

AWS CloudFormation を使用すると、 AWS インフラストラクチャのデプロイを予測どおりに繰り返し作成およびプロビジョニングできます。詳細については AWS CloudFormation、AWS CloudFormation 「 ユーザーガイド」を参照してください。

AWS CloudFormation スタックを作成するには：

1. AWS CLI 「 ユーザーガイド AWS CLI 」の手順に従って、 をインストールして設定します。
2. プロジェクトフォルダのルートディレクトリで、`setup.yaml` という名前のファイルを作成し、それにこの GitHub にコンテンツをコピーします。

> **Note**
>
> AWS CloudFormation テンプレートは、 AWS CDK GitHub で利用可能な を使用して生成されました。の詳細については AWS CDK、 AWS Cloud Development Kit (AWS CDK) デベロッパーガイドを参照してください。

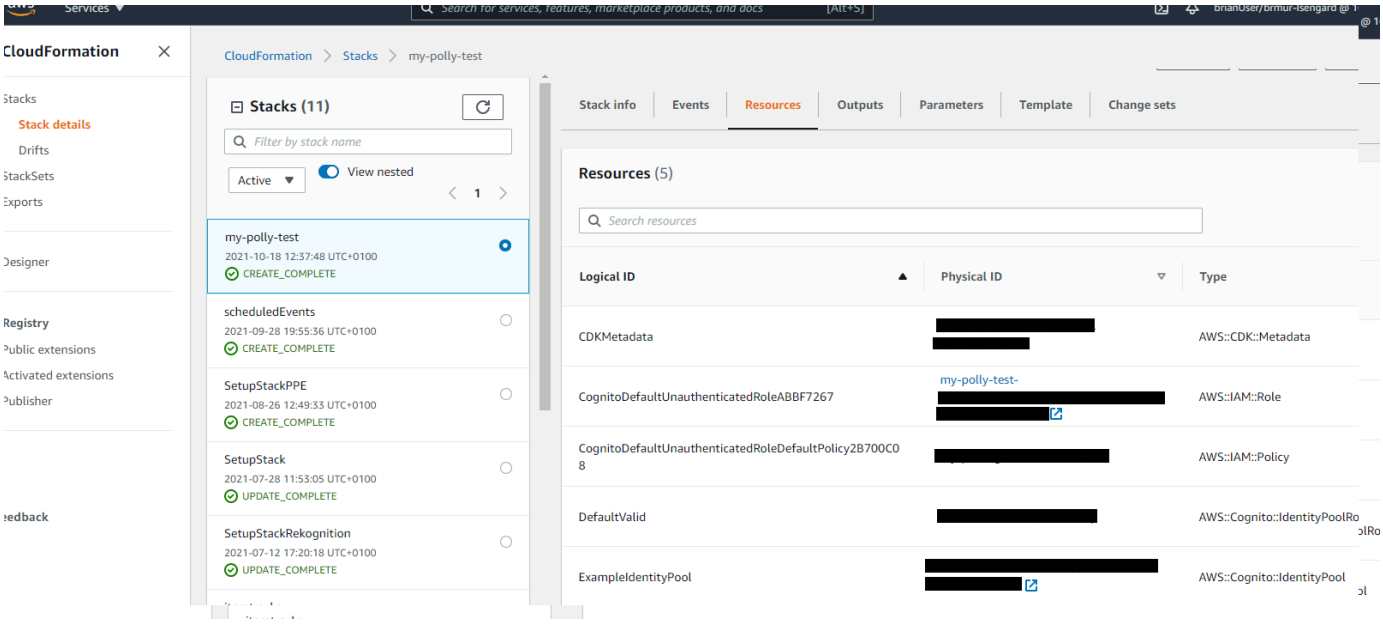
3. コマンドラインから以下のコマンドを実行し、「*STACK_NAME*」をスタックの一意の名前に置き換えます。

> **Important**
>
> スタック名は、 AWS リージョンと AWS アカウント内で一意である必要があります。最大 128 文字まで指定でき、数字とハイフンを使用できます。

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

`create-stack` コマンドパラメータの詳細については、 AWS CLI Command Reference guide (コマンドリファレンスガイド) および「AWS CloudFormation ユーザーガイド」を参照してください。

4. AWS CloudFormation マネジメントコンソールに移動し、スタックを選択し、スタック名を選択し、リソースタブを選択して、作成されたリソースのリストを表示します。



Amazon Polly を使用して録音した音声をAmazon S3 にアップロードします

polly_synthesize_to_s3.jsファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。コードに、*REGION* 、および*BUCKET_NAME*を入力します。Amazon Pollyにアクセスするには、Pollyのクライアントのサービスオブジェクトを作成します。*#IDENTITY_POOL_ID#*を、この例で作成したAmazon CognitoIDプールのサンプルページからIdentityPoolIdを置き換えます。これは、各クライアントオブジェクトにも渡されます。

Amazon PollyクライアントサービスオブジェクトのStartSpeechSynthesisCommandメソッドを呼び出して音声メッセージを合成し、AmazonS3バケットにアップロードします。

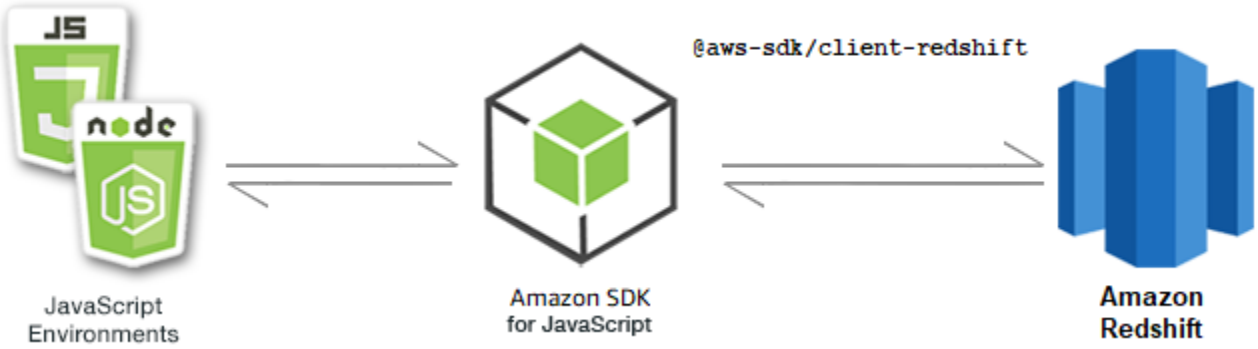
```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "./libs/pollyClient.js";

// 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();
```

このサンプルコードは、このGitHubにあります。

Amazon Redshiftの例

Amazon Redshift は、 クラウド内でのフルマネージド型、ペタバイト規模のデータウェアハウスサービスです。Amazon Redshift データウェアハウスは、ノードと呼ばれるコンピューティングリソースの集合で、クラスターと呼ばれるグループに編成されています。各クラスターは Amazon Redshift エンジンを実行し、1 つ以上のデータベースを含みます。



JavaScript API for Amazon Redshift はAmazon Redshift クライアントクラスを介して公開されます。

トピック

- Amazon Redshiftの例

Amazon Redshiftの例

この例では、一連の Node.js モジュールを使用して、パラメーターの作成、変更、記述をします。次の `Redshift`クライアントクラス方法を使ってAmazon Redshift クラスターを削除します。

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

Amazon Redshift ユーザーの詳細については、「 Amazon Redshift getting started guide 」を参照下さい。

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> - CommonJS 構文を使用する場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

Amazon Redshift クラスターを作成します。

この例では AWS SDK for JavaScriptを使用して Amazon Redshift クラスターを作成する方法を示しています。詳細については、「CreateCluster 」を参照してください。

> **Important**
>
> ここで作成するクラスターはライブです (サンドボックスで実行されるわけではありません)。クラスターを削除するまで、そのクラスターについて Amazon Redshift 標準使用料が発生します。クラスターを作成したときと同じ設定のクラスターを削除すれば、課金される合計金額は最小限になります。

libsディレクトリを作成し、ファイル名redshiftClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon Redshift クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

redshift-create-cluster.jsファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。パラメータオブジェクトを作成し、プロビジョニングするノードタイプ、およびクラスターに自動的に作成されるデータベースインスタンスのマスターサインイン認証情報、最後にクラスタタイプを指定します。

> **Note**
>
> *CLUSTER_NAME*をクラスターの名前に置換します。*[NODE_TYPE]*は、たとえば、'dc2.large' など、プロビジョニングするノードタイプを指定します。*MASTER_USERNAME* そして *MASTER_USER_PASSWORD* は、クラスターの DB インスタンスのマスターユーザーのサインイン認証情報です。*CLUSTER_TYPE*では、クラスターのタイプを入力します。single-

nodeを指定した場合、 NumberOfNodesパラメータは必要ありません。残りのパラメータはオプションです。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにあります。

Amazon Redshift クラスターを変更する

この例では、 AWS SDK for JavaScriptを使用して Amazon Redshift クラスターのマスターユーザーパスワードを変更する方法を示します。その他の設定を変更できる詳細については、「[ModifyCluster]」を参照してください。

libsディレクトリを作成し、ファイル名redshiftClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon Redshift クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

redshift-modify-cluster.jsファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。 AWS リージョン、変更するクラスターの名前、新しいマスターユーザーパスワードを指定します。

> **Note**
>
> *[CLUSTER_NAME]*をクラスターの名前、*[MASTER_USER_PASSWORD]*を新しいマスターユーザーパスワードに置換してください。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにあります。

Amazon Redshift クラスターの詳細を表示します

この例では、 AWS SDK for JavaScriptを使用してAmazon Redshift クラスターの詳細を表示する方法を示しています。オプションの詳細については、「[DescribeClusters]」を参照してください。

libsディレクトリを作成し、ファイル名redshiftClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon Redshift クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

redshift-describe-clusters.jsファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。 AWS リージョン、変更するクラスターの名前、新しいマスターユーザーパスワードを指定します。

> **Note**
>
> *CLUSTER_NAME*をクラスターの名前に置換します。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにあります。

Amazon Redshift クラスターを削除します

この例では、 AWS SDK for JavaScriptを使用してAmazon Redshift クラスターの詳細を表示する方法を示しています。その他の設定を変更できる詳細については、「DeleteCluster」を参照してください。

libsディレクトリを作成し、ファイル名redshiftClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon Redshift クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubに で見つけられます。

redshift-delete-clusters.jsというファイル名で Node.js モジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。 AWS リージョン、変更するクラスターの名前、新しいマスターユーザーパスワードを指定します。削除前にクラスターの最終スナップショットを保存したい場合、そうする場合は、スナップショットのID を指定します。

> **Note**
>
> *CLUSTER_NAME*をクラスターの名前に置換します。*[SkipFinalClusterSnapshot]*で、削除する前に、クラスターの最後のスナップショットを作成するかどうかを指定します。'false' を指定した場合は、*[CLUSTER_SNAPSHOT_ID]*で最後のクラスタースナップショットのidを指定します。このIDは、[Clusters]ダッシュボードのクラスターで[Snapshots]のクラスタ列にあるリンクをクリックして、そして[Snapshots]ペインまでスクロールします。rs:ステムはスナップショットID の一部でないことに注意してください。

```
// 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();
```

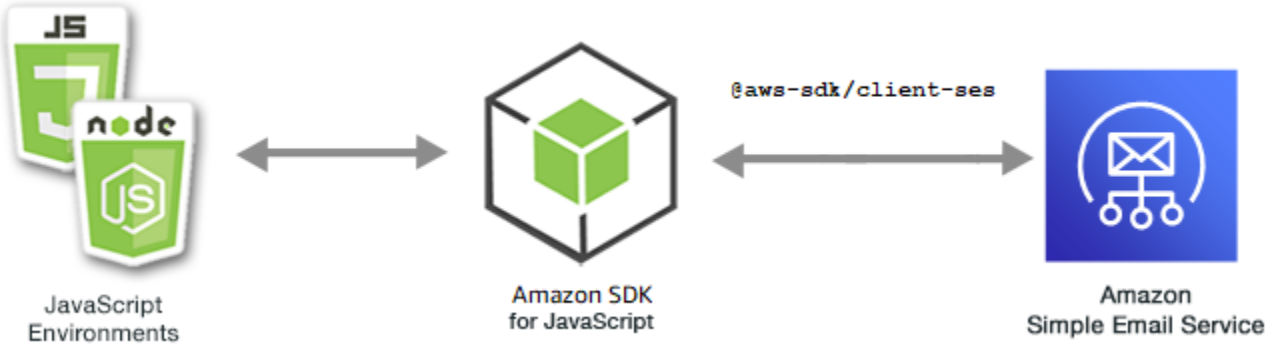
この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにあります。

Amazon Simple Email Servicesの例

Amazon Simple Email Service (Amazon SES) は、デジタルマーケティング担当者やアプリケーションデベロッパーがマーケティング、通知、トランザクションに関する E メールを送信できるように設計された、クラウドベースの E メール送信サービスです。E メールを利用してお客様とのつながりを維持するあらゆる規模の企業を対象とした、コスト効率の高い信頼できるサービスです。



JavaScript API for Amazon SES は SES クライアントクラスを通じて公開されます。Amazon SES クライアントクラスの使用についての詳細は、API リファレンスのClass: SES (クラス : SES)を参照してください。

トピック

- Amazon SES アイデンティティの管理
- Amazon SESでのE メールテンプレートの操作
- Amazon SES を使用してEメールを送信します

Amazon SES アイデンティティの管理

この Node.js コード例は以下を示しています。

- Amazon SES で使用されている E メールアドレスとドメインを確認する方法。
- Amazon SES ID に AWS Identity and Access Management (IAM) ポリシーを割り当てる方法。Amazon SES
- AWS アカウントのすべての Amazon SES ID を一覧表示する方法。
- Amazon SES で使用されているアイデンティティを削除する方法。

Amazon SES アイデンティティは、Amazon SES が E メールの送信に使用する E メールアドレスまたはドメインです。Amazon SES では、E メールアイデンティティを検証して、それを所有していることを確認し、他のユーザーに使用されないようにする必要があります。

Amazon SES の E メールアドレスとドメインを確認する方法の詳細については、Amazon Simple Email Service デベロッパーガイドのAmazon SESでのEメールアドレスとドメインの検証を参照してください。Amazon SES での送信認可の詳細については、Amazon SES 送信認可の概要を参照してください。

シナリオ

この例では、一連の Node.js モジュールを使用して Amazon SES のアイデンティティを検証および管理します。Node.js モジュールは、SES クライアントクラスの次のメソッドを使用し、SDK for JavaScript を使用して E メールアドレスとドメインを検証します。

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

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> - CommonJS 構文を使用する場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

アイデンティティの一覧表示

この例では、Node.js モジュールを使用して Amazon SES で使用する E メールアドレスとドメインを一覧表示します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

ses_listidentities.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

SES クライアントクラスの ListIdentitiesCommand メソッドに IdentityType とその他のパラメータを渡すオブジェクトを作成します。ListIdentitiesCommandメソッドを呼び出すには、Amazon SES サービスオブジェクトを起動し、パラメータオブジェクトを渡します。

返されるdataには、IdentityTypeパラメーターで指定されたドメインIDの配列が含まれます。

> **Note**
>
> *IdentityType* をEmailAddress」または「Domain」の ID タイプに置き換えます。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node ses_listidentities.js
```

このサンプルコードは、このGitHubにで見つけられます。

E メールアドレスアイデンティの検証

この例では、Node.js モジュールを使用して Amazon SES で使用する E メールの送信者を検証します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

ses_verifyemailidentity.jsというファイル名で Node.js モジュールを作成します。前述のように、必要なクライアントとパッケージのダウンロードを含め、SDKを設定します。

SES クライアントクラスの VerifyEmailIdentityCommand メソッドに EmailAddress パラメータを渡すオブジェクトを作成します。VerifyEmailIdentityCommandメソッドを呼び出すには、Amazon SES クライアントサービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> *EMAIL_ADDRESS* を name@example.com などの E メールアドレスに置き換えます。

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

この例を実行するには、コマンドプロンプトで以下を入力します。検証のためにドメインが Amazon SES に追加されます。

```
node ses_verifyemailidentity.js
```

このサンプルコードは、このGitHubにで見つけられます。

ドメイン ID の検証

この例では、Node.js モジュールを使用して Amazon SES で使用する E メールのドメインを検証します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

ses_verifydomainidentity.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

SES クライアントクラスの VerifyDomainIdentityCommand メソッドに Domain パラメータを渡すオブジェクトを作成します。VerifyDomainIdentityCommandメソッドを呼び出すには、Amazon SES クライアントサービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変

更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *DOMAIN_NAME* をドメイン名に置き換えます。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。検証のためにドメインが Amazon SES に追加されます。

```
node ses_verifydomainidentity.js
```

このサンプルコードは、このGitHubにで見つけられます。

アイデンティティの削除

この例では、Node.js モジュールを使用して、Amazon SES で使用されている E メールアドレスまたはドメインを削除します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

ses_deleteidentity.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

SES クライアントクラスの DeleteIdentityCommand メソッドに Identity パラメータを渡すオブジェクトを作成します。DeleteIdentityCommand メソッドを呼び出すには、Amazon SES クライアントサービスオブジェクトを起動するためのrequestを作成し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *IDENTITY_EMAIL* を削除する ID の E メールに置き換えます。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node ses_deleteidentity.js
```

このサンプルコードは、このGitHubにで見つけられます。

Amazon SESでのE メールテンプレートの操作

この Node.js コード例は以下を示しています。

- すべてのEメールテンプレートの一覧表を取得する方法
- E メールテンプレートを取得して更新する方法
- E メールテンプレートを作成して削除する方法

Amazon SES では、Eメールテンプレートを使用してパーソナライズされたEメールメッセージを送信できます。Amazon SESでEメールテンプレートを作成および使用する方法の詳細については、Amazon シンプル Eメールサービス· ディベロッパーガイドの「Amazon SES APIを使用したパーソナライズされたEメールの送信」を参照してください。

シナリオ

この例では、一連の Node.js モジュールを使用して E メールテンプレートを操作します。Node.js モジュールは SDK for JavaScript を使用し、SES クライアントクラスの次のメソッドを使用して E メールテンプレートを作成して使用します。

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

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。

- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。

- CommonJS 構文を使用する場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

Eメールテンプレートの一覧表示

この例では、Node.js モジュールを使用して Amazon SES で使用する E メールテンプレートを作成します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubに で見つけられます。

ses_listtemplates.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

SES クライアントクラスの ListTemplatesCommand メソッドのパラメータを渡すオブジェクトを作成します。ListTemplatesCommandメソッドを呼び出すには、Amazon SES クライアントサービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。Amazon SES はテンプレートのリストを返します。

```
node ses_listtemplates.js
```

このサンプルコードは、このGitHubに で見つけられます。

E メールテンプレートの取得

この例では、Node.js モジュールを使用して Amazon SES で使用する E メールテンプレートを取得します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubに で見つけられます。

ses_gettemplate.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

SES クライアントクラスの GetTemplateCommand メソッドに TemplateName パラメータを渡すオブジェクトを作成します。GetTemplateCommandメソッドを呼び出すには、Amazon SES クライアントサービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *TEMPLATE_NAME* (テンプレート名)を返すテンプレートの名前に置き換えます。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。Amazon SES はテンプレートの詳細を返します。

```
node ses_gettemplate.js
```

このサンプルコードは、このGitHubにで見つけられます。

Eメールテンプレートの作成

この例では、Node.js モジュールを使用して Amazon SES で使用する E メールテンプレートを作成します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

ses_createtemplate.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

TemplateName、HtmlPart、SubjectPart および TextPart を含む、SES クライアントクラスの CreateTemplateCommand メソッドのパラメータを渡すオブジェクトを作成します。CreateTemplateCommandメソッドを呼び出すには、Amazon SES クライアントサービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *TEMPLATE_NAME* を新しいテンプレートの名前に、*HtmlPart* を E メールの HTML タグ付きコンテンツに、*SubjectPart* を E メールの件名に置き換えます。

```
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;
```

```
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。テンプレートが Amazon SES に追加されます。

```
node ses_createtemplate.js
```

このサンプルコードは、このGitHubにで見つけられます。

E メールテンプレートの更新

この例では、Node.js モジュールを使用して Amazon SES で使用する E メールテンプレートを作成します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

ses_updatetemplate.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

必要な TemplateName パラメータを SES クライアントクラスの UpdateTemplateCommand メソッドに渡して、テンプレートで更新する Template パラメータ値を渡すオブジェクトを作成します。UpdateTemplateCommandメソッドを呼び出すには、Amazon SES サービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変

更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *TEMPLATE_NAME* をテンプレートの名前に置き換え、*HTML_PART* を E メールの HTML タグ付きコンテンツに置き換えます。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。Amazon SES はテンプレートの詳細を返します。

```
node ses_updatetemplate.js
```

このサンプルコードは、このGitHubにで見つけられます。

E メールテンプレートの削除

この例では、Node.js モジュールを使用して Amazon SES で使用する E メールテンプレートを作成します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

ses_deletetemplate.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

必要な TemplateName パラメータを SES クライアントクラスの DeleteTemplateCommand メソッドに渡すオブジェクトを作成します。DeleteTemplateCommandメソッドを呼び出すには、Amazon SES サービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *TEMPLATE_NAME* を削除するテンプレートの名前に置き換えてください。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。Amazon SES はテンプレートの詳細を返します。

```
node ses_deletetemplate.js
```

このサンプルコードは、このGitHubにで見つけられます。

Amazon SES を使用してEメールを送信します

この Node.js コード例は以下を示しています。

- テキストまたは HTML の E メールを送信します。
- E メールテンプレートに基づいて E メールを送信します。
- E メールテンプレートに基づいて一括 E メールを送信します。

Amazon SES API は、E メールメッセージの構成に対する制御の程度に応じて、フォーマット済みと raw の 2 つの異なる方法で E メールを送信できます。詳細については、「Amazon SESAPIを使用したフォーマット済みEメールの送信」および「AmazonSESAPIを使用した生のEメールの送信」を参照してください。

シナリオ

この例では、一連の Node.js モジュールを使用してさまざまな方法で E メールを送信します。Node.js モジュールは SDK for JavaScript を使用し、SES クライアントクラスの次のメソッドを使用して E メールテンプレートを作成して使用します。

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

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。

- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。

- これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
- CommonJS 構文を使用する場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

E メールメッセージの送信要件

Amazon SES は E メールメッセージを作成し、送信するメッセージをすぐにキューに入れます。`SendEmailCommand` メソッドを使用して E メールを送信するには、メッセージが以下の要件を満たしている必要があります。

- 検証済みの E メールアドレスまたはドメインからメッセージを送信する必要があります。検証されていないアドレスまたはドメインを使用して E メールを送信しようとすると、`"Email address not verified"` エラーが発生します。
- アカウントがまだ Amazon SES サンドボックスにある場合は、検証済みのアドレスまたはドメイン、または Amazon SES メールボックスシミュレーターに関連付けられた E メールアドレスにのみ送信できます。詳細については、Amazon Simple Email Service デベロッパーガイドの [Eメールアドレスとドメインの検証] を参照してください。
- 添付ファイルを含むメッセージの合計サイズは 10 MB より小さくなければなりません。
- メッセージには少なくとも 1 つの受信者の E メールアドレスを含める必要があります。受信者アドレスは、To: アドレス、CC: アドレス、または BCC: アドレスのいずれかです。受信者の E メールアドレスが無効な場合 (つまり、`UserName@[SubDomain.]Domain.TopLevelDomain` のフォーマットではない場合)、メッセージに他の有効な受信者が含まれていても、メッセージ全体が拒否されます。
- メッセージには、To:、CC:、BCC: のフィールド全体で 50 人を超える受信者を含めることはできません。それ以上の数のユーザーに E メールメッセージを送信する必要がある場合は、受信者リストを 50 ユーザー以下のグループに分割し、`sendEmail` メソッドを数回呼び出して各グループにメッセージを送信することができます。

E メールを送信する

この例では、Node.js モジュールを使用して Amazon SES で E メールを送信します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubに で見つけられます。

ses_sendemail.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

送信者と受信者のアドレス、件名、プレーンテキストおよび HTML 形式のEメール本文などを含む、送信するEメールを定義するパラメータ値を SESクライアントクラスのSendEmailCommandメソッドに渡すオブジェクトを作成します。SendEmailCommandメソッドを呼び出すには、Amazon SES サービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *toAddress* を E メールを送信するアドレスに置き換え、*fromAddress* を E メールを送信元の E メールアドレスに置き換えます。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。E メールは Amazon SES による送信のためにキューに登録されます。

```
node ses_sendemail.js
```

このサンプルコードは、 [here on GitHub]で見つけられます。

テンプレートを使用した E メールを送信する

この例では、Node.js モジュールを使用して Amazon SES で E メールを送信します。ses_sendtemplatedemail.js というファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

送信者と受信者のアドレス、件名、プレーンテキストおよび HTML 形式の E メール本文など、送信する E メールを定義するパラメータ値を SES クライアントクラスの SendTemplatedEmailCommand メソッドに渡すオブジェクトを作成します。SendTemplatedEmailCommandメソッドを呼び出すには、Amazon SES クライアントサービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *REGION* を自分の AWS リージョンに、*USER* を E メールを送信する名前と E メールアドレスに、*VERIFIED_EMAIL* を E メールを送信する E メールアドレスに、*TEMPLATE_NAME* をテンプレートの名前に置き換えます。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。E メールは Amazon SES による送信のためにキューに登録されます。

```
node ses_sendtemplatedemail.js
```

このサンプルコードは、このGitHubにで見つけられます。

テンプレートを使用した一括Eメールを送信します

この例では、Node.js モジュールを使用して Amazon SES で E メールを送信します。

libsディレクトリを作成し、ファイル名sesClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SES クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

ses_sendbulktemplatedemail.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

送信者と受信者のアドレス、件名、プレーンテキストおよび HTML 形式のEメール本文などを含む、送信するEメールを定義するパラメータ値を SESクライアントクラスのSendBulkTemplatedEmailCommandメソッドに渡すオブジェクトを作成します。SendBulkTemplatedEmailCommandメソッドを呼び出すには、Amazon SES サービスオブジェクトを起動し、パラメータを渡します。

> **Note**
>
> この例では、必要な AWS Service V3 パッケージクライアント、V3 コマンドをインポートして使用し、非同期/待機パターンで sendメソッドを使用します。この例は、代わりに少し変更を加えてV2コマンドで作成できます。詳細については、「v3 コマンドの使用」を参照してください。

> **Note**
>
> *USERS* を E メールを送信する名前と E メールアドレスに、*VERIFIED_EMAIL_1* を E メールを送信する E メールアドレスに、*TEMPLATE_NAME* をテンプレートの名前に置き換えます。

```
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;
  }
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。E メールは Amazon SES による送信のためにキューに登録されます。

```
node ses_sendbulktemplatedemail.js
```

このサンプルコードは、このGitHubにで見つけられます。

Amazon Simple Notification Service の例

Amazon Simple Notification Service (Amazon SNS) は、サブスクライブしているエンドポイントやクライアントへのメッセージ配信や送信を調整、管理するウェブサービスです。

Amazon SNS には、発行者とサブスクライバーという 2 種類のクライアントが存在し、それぞれ生産者と消費者とも呼ばれます。

発行者は、論理アクセスポイントおよび通信チャネルであるトピックにメッセージを作成して送信することで、受信者と非同期的に通信します。サブスクライバー (ウェブサーバー、E メールアドレス、Amazon SQS キュー、 AWS Lambda 関数) は、トピックにサブスクライブするときに、サポー

トされているプロトコル (Amazon SQS、HTTP/S、E メール、SMS AWS Lambda) のいずれかを介してメッセージまたは通知を消費または受信します。

Amazon SNS 用JavaScript API は[Class: SNS]を介して公開されます。

トピック

- Amazon SNS でのトピックの管理
- Amazon SNS でのメッセージの公開
- Amazon SNS でのサブスクリプションの管理
- Amazon SNS の SMS メッセージの送信

Amazon SNS でのトピックの管理

この Node.js コード例は以下を示しています。

- 通知を発行できる Amazon SNS でトピックを作成する方法。
- Amazon SNS で作成されたトピックを削除する方法。
- 利用可能なトピックの一覧を取得する方法。
- トピック属性を取得および設定する方法。

シナリオ

この例では、一連の Node.js モジュールを使用して Amazon SNS トピックを作成、一覧表示、および削除し、トピック属性を処理します。Node.js モジュールは、SNS クライアントクラスの以下のメソッドを使用してトピックを管理するために SDK for JavaScript を使用します。

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

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript およびサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> - CommonJS 構文を使用する場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

トピックの作成

この例では、Node.js モジュールを使用して Amazon SNS トピックを作成します。

`libs`ディレクトリを作成し、ファイル名`snsClient.js`でNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*`REGION`* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

create-topic.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

SNS クライアントクラスの CreateTopicCommand メソッドに新しいトピックの Name を渡すためのオブジェクトを作成します。CreateTopicCommandメソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。返されたdataには、トピックの ARN が含まれています。

> **Note**
>
> *TOPIC_NAME*は、SNS トピックの名前に置換してください。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubに で見つけられます。

トピックの一覧表示

この例では、Node.js モジュールを使用してすべての Amazon SNS トピックを一覧表示します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubに で見つけられます。

list-topics.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

SNS クライアントクラスの ListTopicsCommand メソッドに渡す空のオブジェクトを作成します。ListTopicsCommandメソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。返信されたdataには、トピックの Amazon リソースネーム (ARN)の配列が含まれています。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにあります。

トピックの削除

この例では、Node.js モジュールを使用して Amazon SNS トピックを削除します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

delete-topic.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

SNS クライアントクラスの DeleteTopicCommand メソッドに渡すために、削除するトピックの TopicArn を含むオブジェクトを作成します。DeleteTopicCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *[TOPIC_ARN]*を削除するトピックの Amazon リソースネーム (ARN)に置換してください。

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

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubに で見つけられます。

トピック属性の取得

この例では、Node.js モジュールを使用して Amazon SNS トピックの属性を取得します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

get-topic-attributes.jsというファイル名で Node.js モジュールを作成します。前に示したように SDK を設定します。

SNS クライアントクラスの GetTopicAttributesCommand メソッドに渡すために、削除するトピックの TopicArn を含むオブジェクトを作成します。GetTopicAttributesCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> 返された*[TOPIC_ARN]*をトピックのARN に置換してください。

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

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
```

```
  //     Owner: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubに で見つけられます。

トピック属性の設定

この例では、Node.js モジュールを使用して Amazon SNS トピックの変更可能な属性を設定します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubに で見つけられます。

set-topic-attributes.jsというファイル名で Node.js モジュールを作成します。前に示したように SDK を設定します。

属性を設定するトピックの TopicArn、設定する属性の名前、およびその属性の新しい値など、属性の更新のパラメータを含むオブジェクトを作成します。Policy、DisplayName、および DeliveryPolicy 属性のみ設定できます。SNS クライアントクラスの SetTopicAttributesCommand メソッドにパラメータを渡します。SetTopicAttributesCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *ATTRIBUTE_NAME* を設定している属性の名前で、*TOPIC_ARN* 設定したい属性のトピックの Amazon リソースネーム (ARN)で、 *NEW_ATTRIBUTE_VALUE* を属性の新しい値に置き換えてください。

```
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;
```

```
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにで見つけられます。

Amazon SNS でのメッセージの公開

この Node.js コード例は以下を示しています。

- Amazon SNS トピックにメッセージを発行する方法。

シナリオ

この例では、一連の Node.js モジュールを使用して Amazon SNS からトピックのエンドポイント、E メール、または電話番号にメッセージを発行します。Node.js モジュールは SDK for JavaScript を使用して、SNS クライアントクラスのこのメソッドを使用してメッセージを送信します。

- PublishCommand

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript およびサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> - CommonJS 構文を使用する場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

SNS トピックへのメッセージの発行

この例では、Node.js モジュールを使用して Amazon SNS トピックにメッセージを発行します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubに で見つけられます。

publish-topic.jsというファイル名で Node.js モジュールを作成します。前に示したように SDK を設定します。

メッセージテキストと Amazon SNSトピックの Amazon Resource Name(ARN)を含む、メッセージを発行するためのパラメータを含むオブジェクトを作成します。利用可能な SMS 属性の詳細については、「SetSMSAttributes」を参照してください。

パラメータをPublishCommandクライアントクラスのSNSメソッドに渡します。Amazon SNS クライアントサービスオブジェクトを起動する非同期関数を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *MESSAGE_TEXT* をメッセージテキストで、*TOPIC_ARN*をSNS トピックのARNに置き換えてください。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにで見つけられます。

Amazon SNS でのサブスクリプションの管理

この Node.js コード例は以下を示しています。

- Amazon SNS トピックへのすべてのサブスクリプションを一覧表示する方法。
- E メールアドレス、アプリケーションエンドポイント、または AWS Lambda 関数を Amazon SNS トピックにサブスクライブする方法。
- Amazon SNS トピックのサブスクライブを解除する方法。

シナリオ

この例では、一連の Node.js モジュールを使用して通知メッセージを Amazon SNS トピックに発行します。Node.js モジュールは、SNS クライアントクラスの以下のメソッドを使用してトピックを管理するために SDK for JavaScript を使用します。

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

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript およびサードパーティーモジュールをインストールします。「GitHub」の指示に従います。

- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> - CommonJS 構文を使用する場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

サブスクリプションのトピックへの一覧表示

この例では、Node.js モジュールを使用して Amazon SNS トピックへのすべてのサブスクリプションを一覧表示します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubに で見つけられます。

list-subscriptions-by-topic.jsというファイル名で Node.js モジュールを作成します。前に示したように SDK を設定します。

サブスクリプションを一覧表示するトピックの TopicArn パラメータを含むオブジェクトを作成します。SNS クライアントクラスの ListSubscriptionsByTopicCommand メソッドにパラメー

タを渡します。ListSubscriptionsByTopicCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *TOPIC_ARN* をサブスクリプションを一覧表示したいトピックのAmazon リソースネーム (ARN) に置き換えてください。

```
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;
```

```
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにで見つけられます。

E メールアドレスのトピックへのサブスクライブ

この例では、Node.js モジュールを使用して E メールアドレスをサブスクライブし、Amazon SNS トピックから SMTP E メールメッセージを受信するようにします。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

subscribe-email.jsというファイル名で Node.js モジュールを作成します。前に示したように SDK を設定します。

email プロトコル、サブスクライブするトピックの TopicArn、およびメッセージの Endpoint としての E メールアドレスを指定するための Protocol パラメータを含むオブジェクトを作成します。SNS クライアントクラスの SubscribeCommand メソッドにパラメータを渡します。このトピックの他の例が示すように、渡されたパラメータに使用される値に応じて、subscribe メソッドを使用して Amazon SNS トピックにいくつかの異なるエンドポイントをサブスクライブすることができます。

SubscribeCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *TOPIC_ARN* をトピックの Amazon リソースネーム (ARN) に、*EMAIL_ADDRESS* をサブスクライブする E メールアドレスに置き換えます。

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

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにで見つけられます。

サブスクリプションを確認する

この例では、Node.jsモジュールを使用し、以前の Subscribe アクションでエンドポイントに送信したトークンを検証することによって、メッセージを受信するというエンドポイントの所有者の意思を確認します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

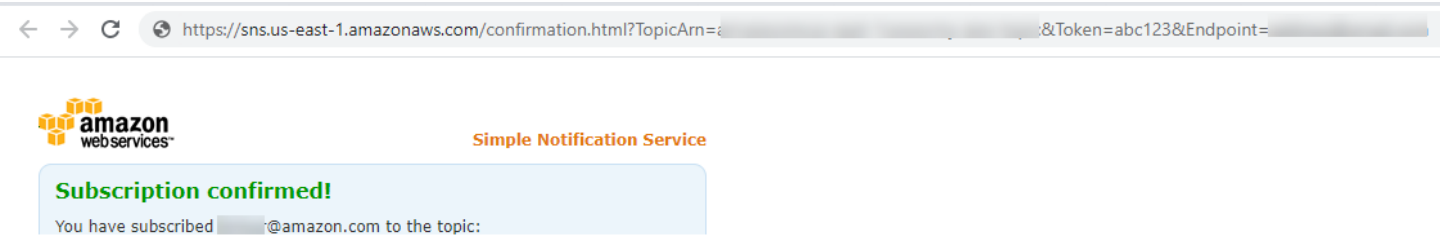
```
import { SNSClient } from "@aws-sdk/client-sns";

// 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({});
```

このサンプルコードは、このGitHubにで見つけられます。

confirm-subscription.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

TOPIC_ARNとTOKENを含むパラメータを定義し、AuthenticateOnUnsubscribeに対してTRUEまたはFALSEの値を定義します。

トークンは、以前のSUBSCRIBEアクションの期間にエンドポイントの所有者に送信される短期間のトークンです。たとえば、電子メールエンドポイントの場合、TOKENは、E メールの所有者に送信されたサブスクリプションの確認メールのURLにあります。例えば、abc123は次のURLのトークンです。



ConfirmSubscriptionCommandメソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *TOPIC_ARN*をトピックの Amazon リソースネーム (ARN)に、*TOKEN*を以前のSubscribeアクションでエンドポイント所有者に送信されたURLのトークン値に置き換えてください。そして、定義*AuthenticateOnUnsubscribe*をTRUEかFALSEの値で定義します。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node confirm-subscription.js
```

このサンプルコードは、このGitHubに で見つけられます。

アプリケーションエンドポイントのトピックへのサブスクライブ

この例では、Node.js モジュールを使用してモバイルアプリケーションのエンドポイントをサブスクライブし、Amazon SNS トピックから通知を受信するようにします。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubに で見つけられます。

subscribe-app.jsというファイル名で Node.js モジュールを作成します。必要なモジュールとパッケージのインストールを含め、前述のようにSDKを設定します。

applicationプロトコルを指定するProtocolパラメータ、サブスクライブするトピックのTopicArn、そしてEndpointパラメータのモバイルアプリケーションエンドポイントの Amazon リソースネーム(ARN)を含むオフジェクトを作成します。SNS クライアントクラスの SubscribeCommand メソッドにパラメータを渡します。

SubscribeCommandメソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *TOPIC_ARN* をトピックのAmazon リソースネーム (ARN)に、 *MOBILE_ENDPOINT_ARN* をトピックスにサブスクライブしているエンドポイントに置き換えてください。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node subscribe-app.js
```

このサンプルコードは、このGitHubにで見つけられます。

Lambda 関数のトピックへのサブスクライブ

この例では、Node.js モジュールを使用して AWS Lambda 関数をサブスクライブし、Amazon SNS トピックから通知を受信します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

subscribe-lambda.jsというファイル名で Node.js モジュールを作成します。前に示したように SDK を設定します。

Protocol パラメータを含むオブジェクトを作成し、 lambda プロトコル、サブスクライブするトピックTopicArnの 、および AWS Lambda 関数の Amazon リソースネーム (ARN) を Endpointパラメータとして指定します。SNS クライアントクラスの SubscribeCommand メソッドにパラメータを渡します。

SubscribeCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *TOPIC_ARN* をトピックの Amazon リソースネーム(ARN)に、*RAMBDA_FUNCTION_ARN*を Lambda 関数のAmazonリソースネーム(ARN)に置き換えてください。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node subscribe-lambda.js
```

このサンプルコードは、このGitHubにで見つけられます。

トピックからのサブスクリプションの解除

この例では、Node.js モジュールを使用して Amazon SNS トピックのサブスクリプションを解除します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

unsubscribe.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。

サブスクリプションを解除するAmazon リソースネーム(ARN)を指定して、SubscriptionArn パラメータを含むオブジェクトを作成します。SNS クライアントクラスの UnsubscribeCommand メソッドにパラメータを渡します。

UnsubscribeCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *TOPIC_SUBSCRIPTION_ARN* をサブスクリプションを解除するAmazon リソースネーム(ARN)に置き換えてください。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node unsubscribe.js
```

このサンプルコードは、このGitHubにで見つけられます。

Amazon SNS の SMS メッセージの送信

この Node.js コード例は以下を示しています。

- Amazon SNS の SMS メッセージングの設定を取得および設定する方法。
- 電話番号をチェックして SMS メッセージの受信をオプトアウトしたかどうかを確認する方法。
- SMS メッセージの受信をオプトアウトした電話番号のリストを取得する方法。
- SMS メッセージを送信する方法。

シナリオ

Amazon SNS を使用して、SMS 対応デバイスにテキストメッセージ (SMS メッセージ) を送信できます。電話番号をトピックにサブスクライブし、トピックへメッセージを送信することにより、電話番号へメッセージを直接送信または、一度に複数の電話番号にメッセージを送信できます。

この例では、一連の Node.js モジュールを使用して、Amazon SNS から SMS 対応デバイスに SMS テキストメッセージを発行します。Node.js モジュールは SDK for JavaScript を使用し、SNS クライアントクラスの以下のメソッドを使用して SMS メッセージを発行します。

- `GetSMSAttributesCommand`
- `SetSMSAttributesCommand`
- `CheckIfPhoneNumberIsOptedOutCommand`
- `ListPhoneNumbersOptedOutCommand`
- `PublishCommand`

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript およびサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **⚠ Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> - CommonJS 構文を使用する場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

SMS 属性の取得

Amazon SNS を使用して、配信の最適化の方法 (コストに対してか、確実な配信に対してか)、毎月の使用量の上限、メッセージ配信がログに記録される方法、SMS の毎日の使用状況レポートをサブスクライブするかどうかなど、SMS メッセージのプリファレンスを指定します。これらのプリファレンスが取得され、Amazon SNS の SMS 属性として設定されます。

この例では、Node.js モジュールを使用して Amazon SNS の現在の SMS 属性を取得します。

`libs`ディレクトリを作成し、ファイル名`snsClient.js`でNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*`REGION`* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

`get-sms-attributes.js`というファイル名で Node.js モジュールを作成します。

前述のように、必要なクライアントとパッケージのダウンロードを含め、SDKを設定します。取得する個々の属性の名前など、SMS 属性を取得するためのパラメータを含むオブジェクトを作成します。利用可能な SMS 属性の詳細については、Amazon Simple Notification Service API リファレンスの SetSMSAttributes を参照してください。

この例では、`DefaultSMSType` 属性を取得します。これは、SMS メッセージが `Promotional` (コストが最も低くなるようにメッセージ配信が最適化されます) として送信されるのか、`Transactional` (信頼性が最も高くなるようにメッセージ配信が最適化されます) として送信されるのかを制御します。SNS クライアントクラスの `SetTopicAttributesCommand` メソッドにパラメータを渡します。`SetSMSAttributesCommand` メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *`ATTRIBUTE_NAME`*を属性の名前に置き換えてください。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node get-sms-attributes.js
```

このサンプルコードは、このGitHubに で見つけられます。

SMS 属性の設定

この例では、Node.js モジュールを使用して Amazon SNS の現在の SMS 属性を取得します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

set-sms-attribute-type.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。設定する個々の属性の名前とそれぞれに設定する値を含む、SMS 属性を設定するためのパラメータを含むオブジェクトを作成します。利用可能な SMS 属性の詳細については、Amazon Simple Notification Service API リファレンスの SetSMSAttributes を参照してください。

この例では、DefaultSMSType 属性を Transactional に設定します。これにより、信頼性が最も高くなるようにメッセージ配信が最適化されます。SNS クライアントクラスの SetTopicAttributesCommand メソッドにパラメータを渡します。SetSMSAttributesCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node set-sms-attribute-type.js
```

このサンプルコードは、このGitHubに で見つけられます。

電話番号がオプトアウトしているかどうかの確認

この例では、Node.js モジュールを使用して電話番号をチェックし、SMS メッセージの受信をオプトアウトしたかどうかを確認します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubに で見つけられます。

check-if-phone-number-is-opted-out.jsというファイル名で Node.js モジュールを作成します。前に示したように SDK を設定します。パラメータとして確認する電話番号を含むオブジェクトを作成します。

この例では、確認する電話番号を指定するために PhoneNumber パラメータを設定します。SNS クライアントクラスの CheckIfPhoneNumberIsOptedOutCommand メソッドにオブジェクトを渡します。CheckIfPhoneNumberIsOptedOutCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

Note

1.

PHONE_NUMBER を電話番号に置き換えてください。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node check-if-phone-number-is-opted-out.js
```

このサンプルコードは、このGitHubに で見つけられます。

オプトアウトした電話番号の一覧表示

この例では、Node.js モジュールを使用して、SMS メッセージの受信からオプトアウトされた電話番号のリストを取得します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubにで見つけられます。

list-phone-numbers-opted-out.jsというファイル名で Node.js モジュールを作成します。前に示したように SDK を設定します。空のオブジェクトをパラメータとして作成します。

SNS クライアントクラスの ListPhoneNumbersOptedOutCommand メソッドにオブジェクトを渡します。ListPhoneNumbersOptedOutCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node list-phone-numbers-opted-out.js
```

このサンプルコードは、このGitHubに で見つけられます。

SMS メッセージの発行

この例では、Node.js モジュールを使用して SMS メッセージを電話番号に送信します。

libsディレクトリを作成し、ファイル名snsClient.jsでNode.js モジュールを作成します。以下のコードをコピーし、ペーストしてAmazon SNS クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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({});
```

このサンプルコードは、このGitHubに で見つけられます。

publish-sms.jsというファイル名で Node.js モジュールを作成します。必要なクライアントとパッケージのインストールを含め、前述のようにSDKを設定します。Message および PhoneNumber パラメータを含むオブジェクトを作成します。

SMS メッセージを送信するときは、E.164 形式を使用して電話番号を指定します。E.164 は、国際的な音声通信に使用される電話番号の構造の規格です。この形式に従う電話番号には最大 15 桁を設定でき、プラス記号 (+) および国コードのプレフィックスがついています。たとえば、E.164 形式の米国の電話番号は +1001XXX5550100 として表示されます。

この例では、メッセージを送信するための電話番号を指定する PhoneNumber パラメータを設定します。SNS クライアントクラスの PublishCommand メソッドにオブジェクトを渡しま

す。PublishCommand メソッドを呼び出すには、Amazon SNS サービスオブジェクトを起動する非同期機能 を作成し、パラメータオブジェクトを渡します。

> **Note**
>
> *TEXT_MESSAGE*をテキストメッセージに、*PHONE_NUMBER*を電話番号に置き換えてください。

```
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;
};
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node publish-sms.js
```

このサンプルコードは、このGitHubにで見つけられます。

Amazon Transcribeの例

Amazon Transcribe を使用すると、開発者はアプリケーションに音声認識機能を簡単に追加できます。

「Amazon Transcribe」(Amazon Transcribe)のJavaScriptAPIは、TranscribeServiceクライアントクラスを介して公開されます。

トピック

- Amazon Transcribeの例
- Amazon Transcribe Medicalの例

Amazon Transcribeの例

この例では、一連のNode.jsモジュールを使用して、TranscribeServiceクライアントクラスの次のメソッドを使用して文字起こしジョブを作成、一覧表示、および削除します。

- StartTranscriptionJobCommand
- ListTranscriptionJobsCommand
- DeleteTranscriptionJobCommand

Amazon Transcribe ユーザーの詳細については、Amazon Transcribe 開発者ガイドを参照してください。

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> - CommonJS 構文を使用する場合は、JavaScript ES6/CommonJS 構文を参照してください。

Amazon Transcribe ジョブを開始します

この例は、 AWS SDK for JavaScriptを使用してAmazon音声文字変換ジョブを開始する方法を示しています。詳細については、StartTranscriptionJobCommandを参照してください。

`libs`ディレクトリを作成し、ファイル名`transcribeClient.js`でNode.js モジュールを作成します。以下のコードをコピーして、ペーストして、Amazon Transcribe クライアントオブジェクトを作成します。*`REGION`* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

transcribe-create-job.jsファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。必要なパラメータを指定してパラメータ オブジェクトを作成します。StartMedicalTranscriptionJobCommandコマンドを使用してジョブを開始します。

> **Note**
>
> *MEDICAL_JOB_NAME*をトランスクリプションジョブの名前に置き換えてください。*OUTPUT_BUCKET_NAME*には、出力が保存されるAmazonS3バケットを指定します。*JOB_TYPE*には、ジョブのタイプを指定します。*SOURCE_LOCATION*には、ソースファイルの場所を指定します。*SOURCE_FILE_LOCATION*には、入力メディアファイルの場所を指定します。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにあります。

AmazonTranscribeジョブを一覧表示します

この例は、 AWS SDK for JavaScriptを使用して「Amazon Transcribe」(Amazon Transcribe)文字起こしジョブを一覧表示する方法を示しています。変更できる他の設定の詳細については、`ListTranscriptionJobCommand`を参照してください。

`libs`ディレクトリを作成し、ファイル名`transcribeClient.js`でNode.js モジュールを作成します。以下のコードをコピーして、ペーストして、Amazon Transcribe クライアントオブジェクトを作成します。*`REGION`* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

`transcribe-list-jobs.js`ファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。必要なパラメータを使用して パラメータオブジェクトを作成します。

> **Note**
>
> *`KEY_WORD`*を、返されるジョブ名が含まれている必要のあるキーワードに置き換えます。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node transcribe-list-jobs.js
```

このサンプルコードは、このGitHubにあります。

Amazon Transcribe ジョブを削除します

この例では、 AWS SDK for JavaScriptを使用してAmazon Transcribe文字起こしジョブを削除する方法を示します。オプションの詳細については、`DeleteTranscriptionJobCommand`を参照してください。

`libs`ディレクトリを作成し、ファイル名`transcribeClient.js`でNode.js モジュールを作成します。以下のコードをコピーして、ペーストして、Amazon Transcribe クライアントオブジェクトを作成します。*`REGION`* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

transcribe-delete-job.jsファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。 AWS リージョンと、削除するジョブの名前を指定します。

> **Note**
>
> *JOB_NAME*を削除するジョブの名前に置き換えてください。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node transcribe-delete-job.js
```

このサンプルコードは、GitHubで公開されています。

Amazon Transcribe Medicalの例

この例では、一連のNode.jsモジュールを使用して、TranscribeServiceクライアントクラスの次のメソッドを使用して、医療文字起こしジョブを作成、一覧表示、および削除します。

- StartMedicalTranscriptionJobCommand
- ListMedicalTranscriptionJobsCommand
- DeleteMedicalTranscriptionJobCommand

Amazon Transcribe ユーザーの詳細については、Amazon Transcribe 開発者ガイドを参照してください。

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> これらの例は、ECMAScript6 (ES6) を使用してクライアントサービスオブジェクトとコマンドをimport/export する方法を示します。
>
> - これには Node.js バージョン13.x以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> - CommonJS 構文を使用する場合は、JavaScript ES6/CommonJS 構文を参照してください。

Amazon Transcribe のメディカル文字起こしジョブを開始します

この例では、 AWS SDK for JavaScriptを使用してAmazon Transcribe のメディカル文字起こしジョブをスタートする方法を示します。詳細については、 startMedicalTranscriptionJob を参照してください。

libsディレクトリを作成し、ファイル名transcribeClient.jsでNode.js モジュールを作成します。以下のコードをコピーして、ペーストして、Amazon Transcribe クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

transcribe-create-medical-job.jsファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。必要なパラメータを指定してパラメータ オブジェクトを作成します。StartMedicalTranscriptionJobCommandコマンドを使用してメディカル ジョブをスタートします

> **Note**
>
> *MEDICAL_JOB_NAME*をメディカル文字起こしジョブの名前に置き換えてください。*OUTPUT_BUCKET_NAME*には、出力が保存されるAmazonS3バケットを指定します。*JOB_TYPE*には、ジョブのタイプを指定します。*SOURCE_LOCATION*には、ソースファイルの場所を指定します。*SOURCE_FILE_LOCATION*には、入力メディアファイルの場所を指定します。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにあります。

Amazon Transcribe メディカルジョブを一覧表示します

この例では、 AWS SDK for JavaScriptを使用して Amazon Transcribe Transcribeジョブを一覧表示する方法を示します。詳細については、ListTranscriptionMedicalJobsCommandを参照してください。

libsディレクトリを作成し、ファイル名transcribeClient.jsでNode.js モジュールを作成します。以下のコードをコピーして、ペーストして、Amazon Transcribe クライアントオブジェクトを作成します。*REGION* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

transcribe-list-medical-jobs.jsファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。必要なパラメータを使用してパラメータオブジェクトを作成し、ListMedicalTranscriptionJobsCommand コマンドを使用してメディカルジョブを一覧表にします。

> **Note**
>
> *KEYWORD*を、返されるジョブ名が含まれている必要のあるキーワードに置き換えます。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

```
node transcribe-list-medical-jobs.js
```

このサンプルコードは、このGitHubにあります。

Amazon Transcribe メディカル ジョブを削除します

この例では、 AWS SDK for JavaScriptを使用して Amazon Transcribe 文字起こしジョブを削除する方法を示します。オプションの詳細については、`DeleteTranscriptionMedicalJobCommand`を参照してください。

libsディレクトリを作成し、ファイル名`transcribeClient.js`でNode.js モジュールを作成します。以下のコードをコピーして、ペーストして、Amazon Transcribe クライアントオブジェクトを作成します。*`REGION`* を自分の AWS リージョンに置き換えます。

```
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 };
```

このサンプルコードは、このGitHubにで見つけられます。

`transcribe-delete-job.js`ファイル名を使用してNode.jsモジュールを作成します。前に示したように、必要なクライアントとパッケージのインストールを含むSDKが設定されていることを確認してください。必要なパラメータを使用してパラメータオブジェクトを作成し、`DeleteMedicalJobCommand`のコマンドを使用してメディカルジョブを削除します。

> **Note**
>
> *`JOB_NAME`*を削除するジョブの名前に置換します。

```
// 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();
```

この例を実行するには、コマンドプロンプトで以下を入力します。

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

このサンプルコードは、このGitHubにあります。

Amazon EC2 インスタンスでの Node.js を設定する

SDK for JavaScript で Node.js を使用するには、通常、Amazon Elastic Compute Cloud (Amazon EC2) インスタンス上で Node.js ウェブアプリケーションをセットアップして実行します。このチュートリアルでは、Linux インスタンスを作成し、SSH を使用してインスタンスに接続してから、そのインスタンスで実行する Node.js をインストールします。

前提条件

このチュートリアルは、インターネットから到達可能で、SSHを使用して接続できるパブリックDNS名でLinuxインスタンスをすでに起動していることを前提としています。これを行う方法の詳細

については、Amazon EC2 ユーザーガイドの「ステップ 1: インスタンスを起動する」を参照してください。

> **Important**
>
> 新しい Amazon EC2 インスタンスを起動するときは、Amazon Linux 2023 用の Amazon マシンイメージ (AMI) を使用します。

また、セキュリティグループを設定して、SSH (ポート 22)、 HTTP (ポート 80)、HTTPS (ポート 443) 接続を有効にしている必要もあります。これらの前提条件の詳細については、Amazon EC2 ユーザーガイド」の「Amazon EC2 でのセットアップ」を参照してください。 Amazon EC2

手順

次の手順により、Amazon Linux インスタンスで Node.js をインストールすることができます。このサーバーを使用して Node.js ウェブアプリケーションをホストすることができます。

Linux インスタンスで Node.js を設定するには

1. SSH を使用して、Linux インスタンスに `ec2-user` として接続します。
2. コマンドラインで次のように入力して、ノードバージョンマネージャー(nvm)をインストールしてください。

   > **Warning**
   >
   > AWS は、次のコードを制御しません。実行する前に、その信頼性と整合性を検証する必要があります。このコードの詳細については、[nvm] GitHub リポジトリで参照できます。

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

   nvmは Node.js の複数のバージョンをインストールすることができ、そして、それらの切り替えもできるため、nvmを使用して Node.js をインストールします。
3. コマンドラインで次のように入力し、nvm をロードします。

   ```
   source ~/.bashrc
   ```

4. コマンドラインで次のように入力し、nvm を使用して Node.js の最新の LTS バージョンをインストールします。

```
nvm install --lts
```

Node.js をインストールすると、Node Package Manager (npm) もインストールされるため、必要に応じて追加のモジュールをインストールできます。

5. コマンドラインで次のように入力して、Node.js が正しくインストールされ、実行されていることをテストします。

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

これにより、実行中の Node.js のバージョンを示す次のメッセージが表示されます。

```
Running Node.js VERSION
```

> **Note**
>
> ノードのインストールは、現在の Amazon EC2 セッションにのみ適用されます。CLI セッションを再開する場合は、nvm を再度使用して、インストールされているノードバージョンを有効にする必要があります。インスタンスが終了したら、ノードを再インストールする必要があります。別の方法として、次のトピックで説明するように、保持したい設定が完了したら Amazon EC2 インスタンスの Amazon マシンイメージ (AMI) を作成します。

Amazon マシンイメージ（AMI）を作成します

Amazon EC2 インスタンスで Node.js をインストールしたら、そのインスタンスから Amazon マシンイメージ (AMI) を作成できます。AMI を作成することで、同じ Node.js のインストールで複数の Amazon EC2 インスタンスを簡単にプロビジョニングできます。既存のインスタンスから AMI を作成する方法の詳細については、Amazon EC2 ユーザーガイド」の「Amazon EBS-backed Linux AMI の作成」を参照してください。

関連リソース

このトピックで使用されているコマンドおよびソフトウェアの詳細については、次のウェブページを参照してください。

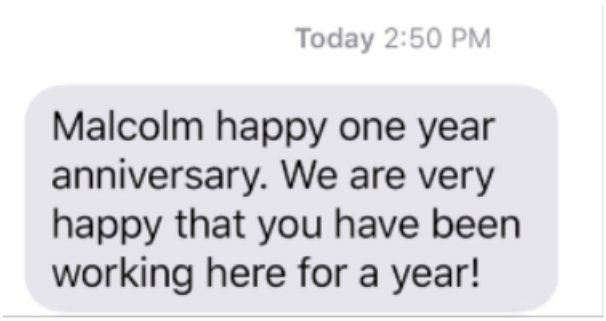
- ノードバージョンマネージャー (nvm)- [nvm repo on GitHub]を参照してください。
- ノードパッケージマネージャー (npm)-[npm website]を参照してください。

API Gateway を使用した Lambdaを呼び出し

Lambda 関数を呼び出すには、Amazon API Gateway を使用します。Amazon API Gateway は、REST、HTTP、WebSocket APIsに作成、公開、維持、モニタリング、保護するための AWS サービスです。API デベロッパーは、 AWS または他のウェブサービス、および AWS クラウドに保存されているデータにアクセスする APIs を作成できます。API Gateway デベロッパーとして、独自のクライアントアプリケーションで使用するためのAPIを作成できます。詳細については、[What is Amazon API Gateway](Amazon API Gatewayとは)を参照してください。

AWS Lambda は、サーバーのプロビジョニングや管理を行わずにコードを実行できるようにするコンピューティングサービスです。Lambda 関数は、さまざまなプログラミング言語で作成できます。詳細については AWS Lambda、「 とは AWS Lambda」を参照してください。

この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。例えば、次の図に示すように、組織が 1 周年記念日に従業員を祝福するモバイルテキストメッセージを送信するとします。



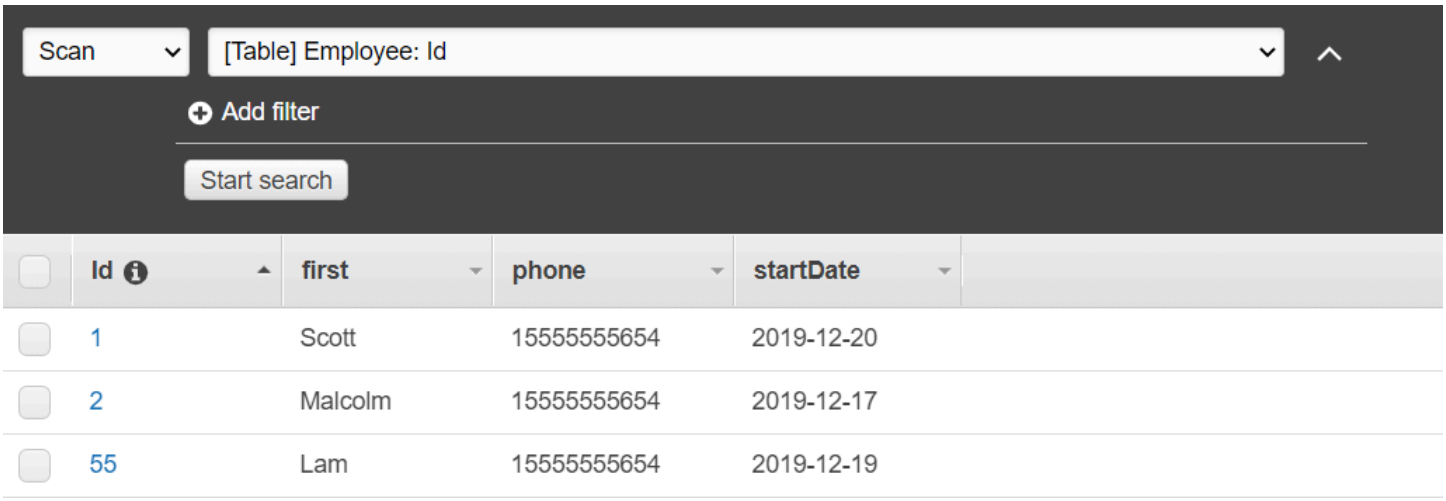
この例は完了までに約 20 分かかります。

この例では、JavaScript ロジックを使用して、このユースケースを実行するソリューションを作成する方法を示しています。例えば、データベースを読み取り、1 年記念日になった従業員を特定する方法、データを処理する方法、およびテキストメッセージを送信する方法について全てLambda 関数を使用して説明します。次に、API Gateway を使用して Rest エンドポイントを使用してこの AWS Lambda 関数を呼び出す方法について説明します。例えば、この curl コマンドを使用して Lambda 関数を呼び出すことができます。:

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/
employee"
```

この AWS チュートリアルでは、これらのフィールドを含む Employee という名前の Amazon DynamoDB テーブルを使用します。

- id - 表のプライマリキー。
- 名前 - 従業員のファーストネーム。
- 電話 - 従業員の電話番号。
- 開始日 - 従業員の入社日。



Important

完了コスト: このドキュメントに含まれる AWS サービスは、 AWS 無料利用枠に含まれています。ただし、この例を完了したら必ずすべてのリソースを終了して料金が発生しないようにしてください。

アプリケーションを構築するには、

1. 前提条件を満たします
2. AWS リソースを作成する
3. ブラウザスクリプトを準備
4. Lambda 関数の作成とアップロード

5. Lambda 関数をデプロイします
6. アプリを実行
7. リソースを削除します

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

AWS リソースを作成する

このチュートリアルでは、以下のリソースが必要です。

- `Id` という名前のキーと前の図に示されているフィールドを持つ `Employee` という Amazon DynamoDB テーブル。このユースケースでテストする有効な携帯番号を含め、正しいデータを入力してください。詳細については、テーブルの作成を参照してください。
- Lambda関数を実行するためのアクセス許可が付与されたIAMロール。
- Lambda 関数をホストするAmazon S3 バケット。

これらのリソースは手動で作成できますが、このチュートリアルで説明されている AWS CloudFormation ように、 を使用してこれらのリソースをプロビジョニングすることをお勧めします。

を使用して AWS リソースを作成する AWS CloudFormation

AWS CloudFormation を使用すると、 AWS インフラストラクチャのデプロイを予測どおりに繰り返し作成およびプロビジョニングできます。詳細については AWS CloudFormation、AWS CloudFormation 「 ユーザーガイド」を参照してください。

を使用して AWS CloudFormation スタックを作成するには AWS CLI :

1. AWS CLI 「 ユーザーガイド」の手順に従って AWS CLI 、 をインストールして設定します。
2. プロジェクトフォルダのルートディレクトリで、setup.yaml という名前のファイルを作成し、それにこの GitHub にコンテンツをコピーします。

> **Note**
>
> AWS CloudFormation テンプレートは、 AWS CDK GitHub で利用可能な を使用して生成されました。の詳細については AWS CDK、 AWS Cloud Development Kit (AWS CDK) デベロッパーガイドを参照してください。

3. コマンドラインから以下のコマンドを実行し、「*STACK_NAME*」をスタックの一意の名前に置き換えます。

> **Important**
>
> スタック名は、 AWS リージョンと AWS アカウント内で一意である必要があります。最大 128 文字まで指定でき、数字とハイフンを使用できます。

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

create-stack コマンドパラメータの詳細については、 AWS CLI Command Reference guide (コマンドリファレンスガイド) および「AWS CloudFormation ユーザーガイド」を参照してください。

4. 次に、表に入力します の手順に従ってテーブルに入力します。

表に入力します

テーブルにデータを入力するには、まず libs という名前のディレクトリを作成し、そこに dynamoClient.js という名前のファイルを作成し、それに以下の内容を貼り付けます。

```
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 };
```

このコードはこのGitHubで利用できます。

次に、populate-table.js というファイルをプロジェクトフォルダのルートディレクトリに作成し、この GitHub にコンテンツをコピーします。項目の1つについて、phone のプロパティの値を E.164形式の有効な携帯電話番号に置き換え、startDate の値を今日の日付に置き換えます。

コマンドラインから、以下のコマンドを実行します。

```
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();
```

このコードはこのGitHubで利用できます。

AWS Lambda 関数の作成

SDK の設定

libs のディレクトリで snsClient.js と lambdaClient.js という名前のファイルを作成し、これらのファイルに以下の内容をそれぞれ貼り付けます。

```
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 };
```

REGION を AWS リージョンに置き換えます。このコードはこのGitHubにで利用できます。

```
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 };
```

REGION を AWS リージョンに置き換えます。このコードはこのGitHubにで利用できます。

まず、必要な AWS SDK for JavaScript (v3) モジュールとコマンドをインポートします。次に、今日の日付を計算し、パラメータに割り当てます。3 番目に、ScanCommand のパラメータを作成します。*TABLE_NAME* を、この例の「AWS リソースを作成する 」セクションで作成したテーブルの名前に置き換えます。

以下のコードスニペットは、このステップを示しています (詳細な例については、Lambda 関数をバンドルします を参照してください)。

```
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",
};
```

DynamoDB テーブルをスキャンします

まず、Amazon SNS PublishCommand を使用してテキストメッセージを公開するために sendText と呼ばれる非同期/待機関数を作成します。次に、今日が勤務記念日である従業員の DynamoDB テーブルをスキャンし、sendText 関数を呼び出してこれらの従業員にテキストメッセージを送信する try ブロックパターンを追加します。エラーが発生した場合は、catch ブロックされます。

以下のコードスニペットは、このステップを示しています (詳細な例については、Lambda 関数をバンドルします を参照してください)。

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

Lambda 関数をバンドルします

このトピックでは、 mylambdafunction.tsと、この例に必要な AWS SDK for JavaScript モジュールを というバンドルされたファイルにバンドルする方法について説明しますindex.js。

1. まだの場合は、この例の前提条件タスクに従ってwebpackをインストールしてください。

> **Note**
>
> Webpack の詳細については、「アプリケーションを webpack にバンドルする」を参照してください。

2. コマンドラインで以下を実行して、この例の JavaScript を <index.js> というファイルにバンドルします。

```
webpack mylambdafunction.ts --mode development --target node --devtool false --
output-library-target umd -o index.js
```

> **Important**
>
> 出力の名前がindex.jsであることに注意してください。Lambda関数が機能するにはindex.jsハンドラーが必要です。

3. バンドルされた出力ファイル index.js を、mylambdafunction.zipという名前の ZIP ファイルに圧縮します。
4. このチュートリアルのAWS リソースを作成する トピックで作成したAmazonS3バケットにmylambdafunction.zipをアップロードします。

Lambda 関数をデプロイします

プロジェクトのルートで、lambda-function-setup.ts ファイルを作成し、それに以下の内容をペーストします。

BUCKET_NAME を Lambda 関数の ZIP バージョンをアップロードした Amazon S3 バケットの名前に置き換えます。*ZIP_FILE_NAME*を、Lambda関数のZIPバージョンの名前に置き換えます。*ROLE*をこのチュートリアルのAWS リソースを作成する トピックで作成した IAM ロールのAmazonリソースナンバー (ARN)に置き換えます。*LAMBDA_FUNCTION_NAME*をLambda関数名に置き換えます。

```
// 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();
```

コマンドラインで次を入力して、Lambda 関数をデプロイします。

```
node lambda-function-setup.ts
```

このコード例はこのGitHubにで利用可能です。

Lambda 関数を呼び出すために API Gatewayを設定します

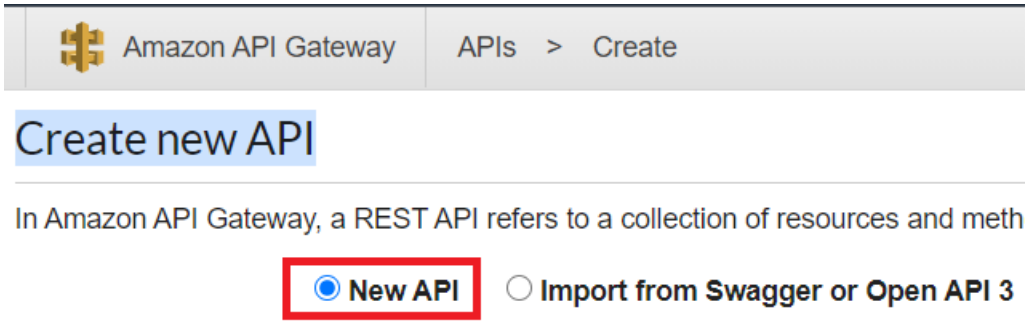
アプリを構築するには

1. rest API を作成する
2. API Gateway メソッドをテストする
3. API Gateway メソッドをデプロイする

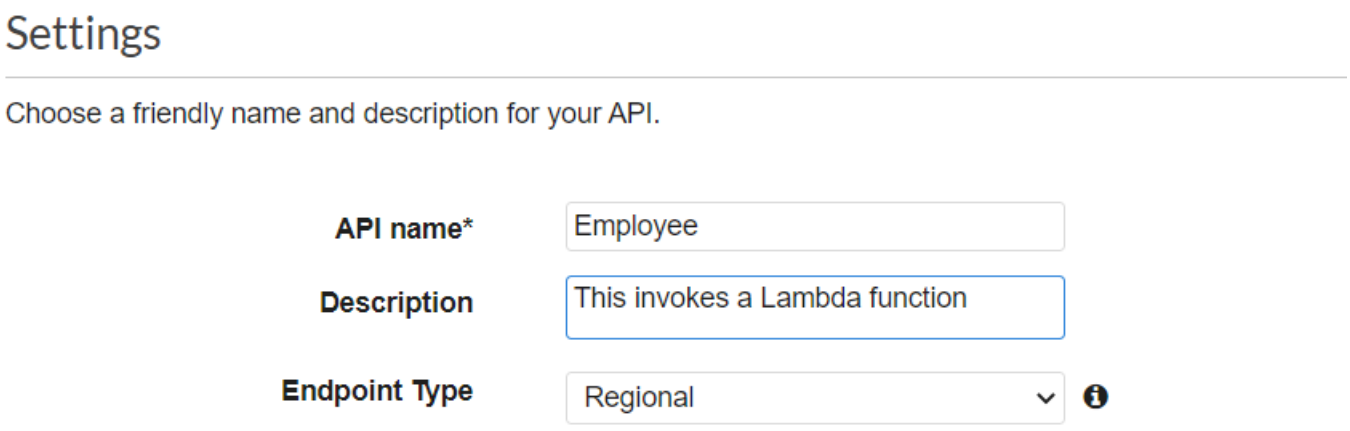
rest API を作成する

API Gateway コンソールを使用して、Lambda 関数のrestエンドポイントを作成できます。完了したら、restful 呼び出しを使用して Lambda 関数を呼び出すことができます。

1. [Amazon API Gateway console](Amazon API Gateway コンソール)にサインインします。
2. REST APIで、 [Build] (構築) を選択します。
3. [New API] (新規API)を選択します。

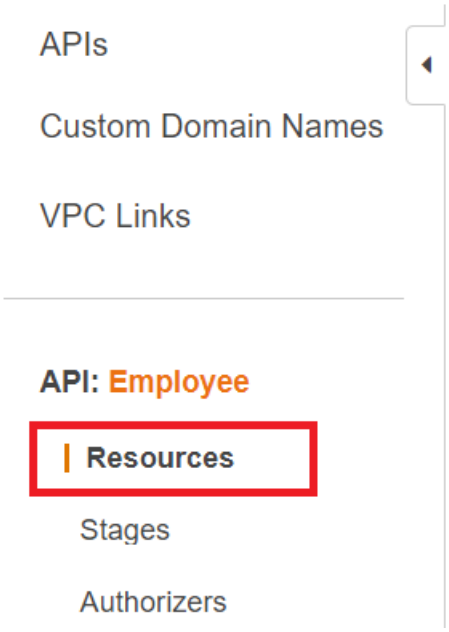


4. [Employee]をAPI 名として指定し、説明を入力します。

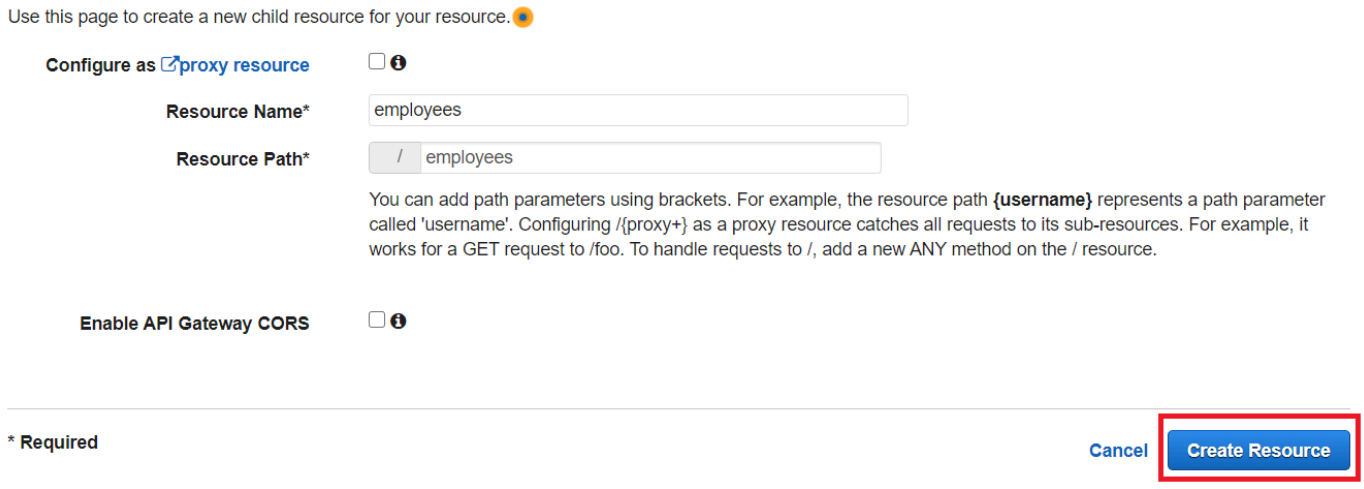


5. API の作成 を選択します。

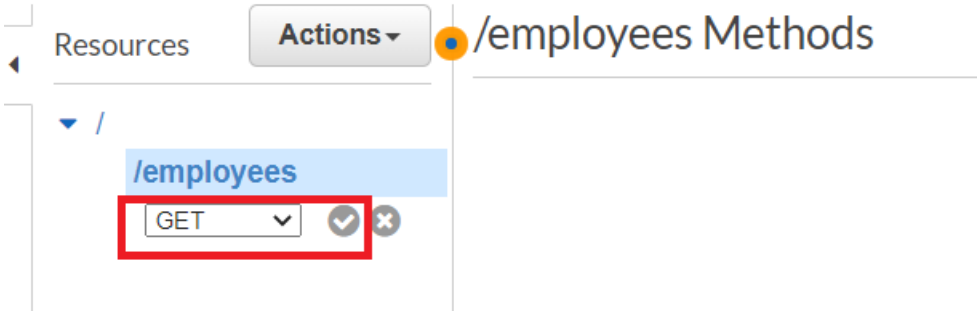
6. Employeeセクションの[Resources]を選択します。



7. 名前フィールドの employeesを指定します。
8. [Create Resources] (リソースの作成) を選択します。
9. [Actions] (アクション)のドロップダウンから [Create Resource] (リソースの作成)を選択します。



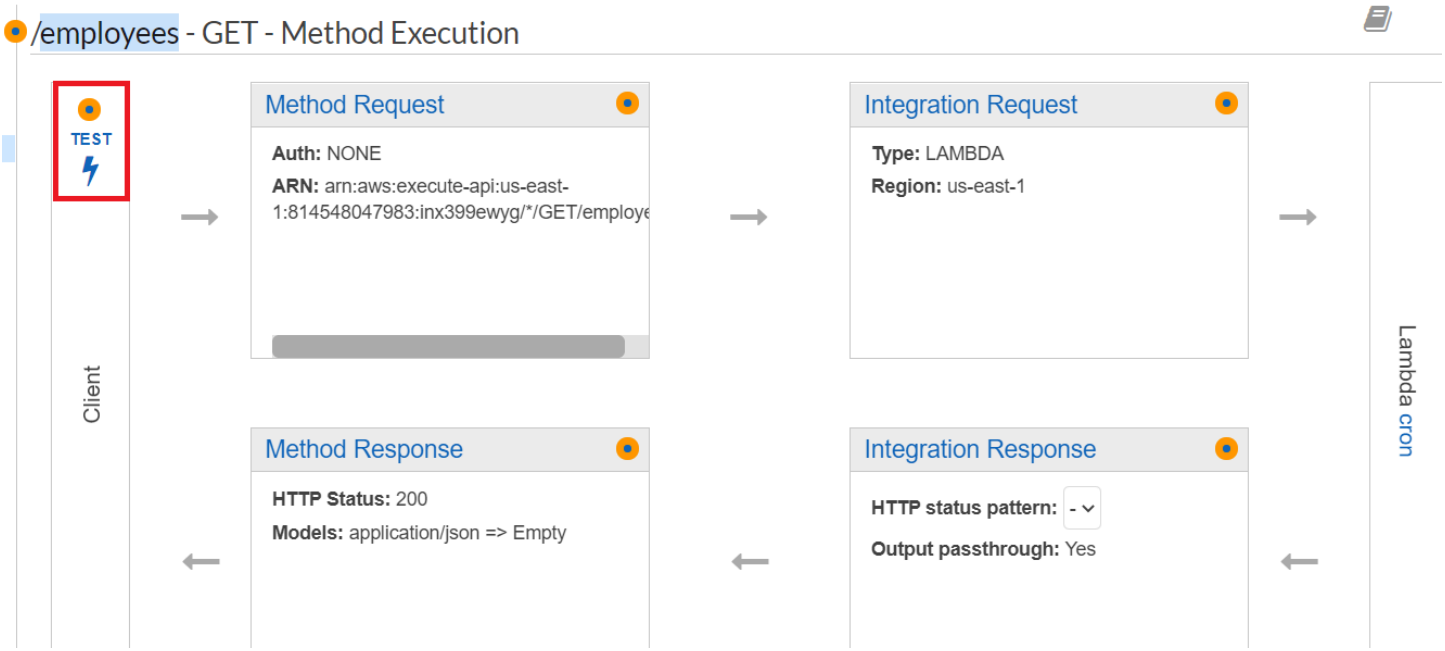
10. [/employees]を選択し、[Create Method] から[Actions]を選択し、[GET]を[/employees]下のドロップダウンメニューから選択します。チェックマークアイコンを選択します。



11. Lambda functionを選択し、Lambda 関数名としてmylambdafunctionと入力します。[Save] を選択します。

API Gateway メソッドをテストする

チュートリアルのこの時点で、mylambdafunctionのLambda 関数を呼び出す API Gateway メソッドをテストできます。メソッドをテストするには、次の図に示す[Test]を選びます。

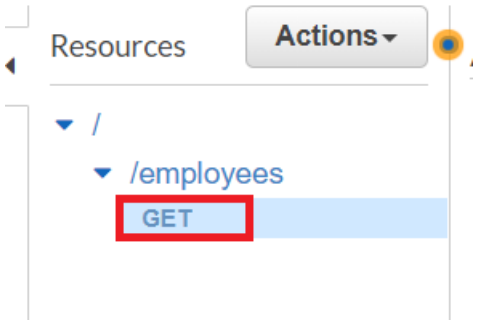


Lambda 関数が呼び出されると、ログファイルを表示して成功したメッセージを表示できます。

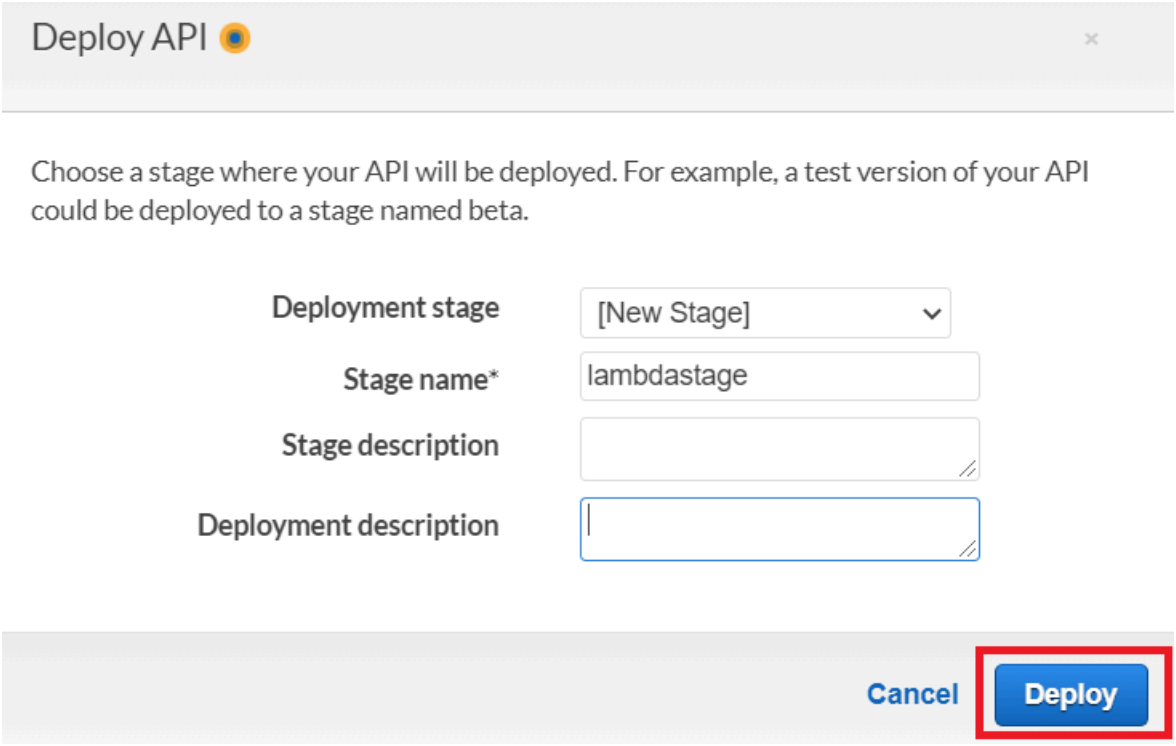
API Gateway メソッドをデプロイする

テストが成功したら、 Amazon API Gateway コンソールから、メソッドをデプロイできます。

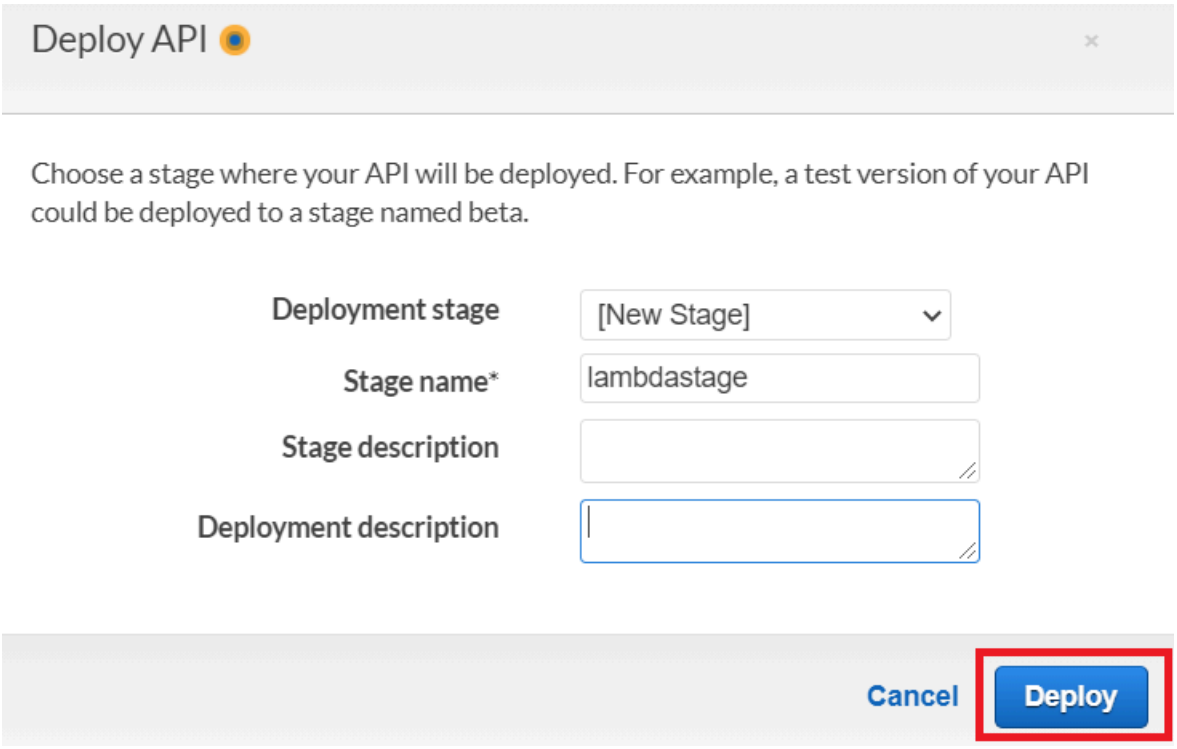
1. [GET] (取得する) を選択します。



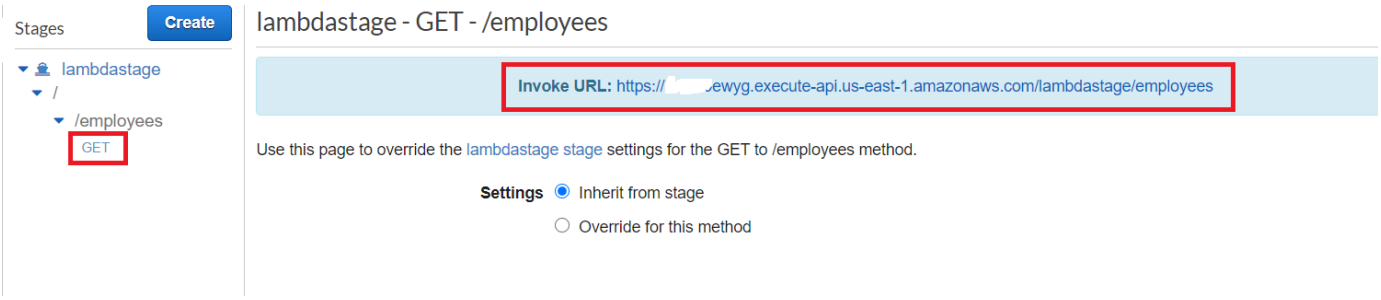
2. [Actions]（アクション）ドロップダウンから[Deploy API](デプロイAPI ）を選択します。



3. [Deploy API]フォームに入力し、[Deploy]を選択します。



4. [Save Changes] を選択します。
5. Getをもう一度選択し、URL が変更されることに注意します。これは、Lambda 関数の呼び出しに使用できるURLです。



リソースを削除します

お疲れ様でした。 AWS SDK for JavaScriptを使用してAmazon API Gateway を介しLambda 関数を呼び出します。このチュートリアルの冒頭で説明したように、このチュートリアルを進めたうえで、作成したすべてのリソースを終了して、料金が発生しないようにしてください。これを行うには、このチュートリアルの AWS リソースを作成する トピックで作成した AWS CloudFormation スタックを次のように削除します。

1. AWS CloudFormationAWS マネジメントコンソールで を開きます。

2. 「スタック」ページを開き、スタックを選択します。
3. [削除] を選択します。

AWS Lambda 関数を実行するためのスケジュールされたイベントの作成

Amazon CloudWatch イベントを使用して、 AWS Lambda 関数を呼び出すスケジュールされたイベントを作成できます。cron式を使用してLambda関数が呼び出されるタイミングをスケジュールするようにCloudWatchイベントを設定できます。例えば、CloudWatchイベントをスケジュールして、平日に毎日Lambda関数を呼び出すことができます。

AWS Lambda は、サーバーのプロビジョニングや管理を行わずにコードを実行できるようにするコンピューティングサービスです。Lambda 関数は、さまざまなプログラミング言語で作成できます。詳細については AWS Lambda、「とは AWS Lambda」を参照してください。

このチュートリアルでは、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。例えば、次の図に示すように、組織が 1 周年記念日に従業員を祝福するモバイルテキストメッセージを送信するとします。

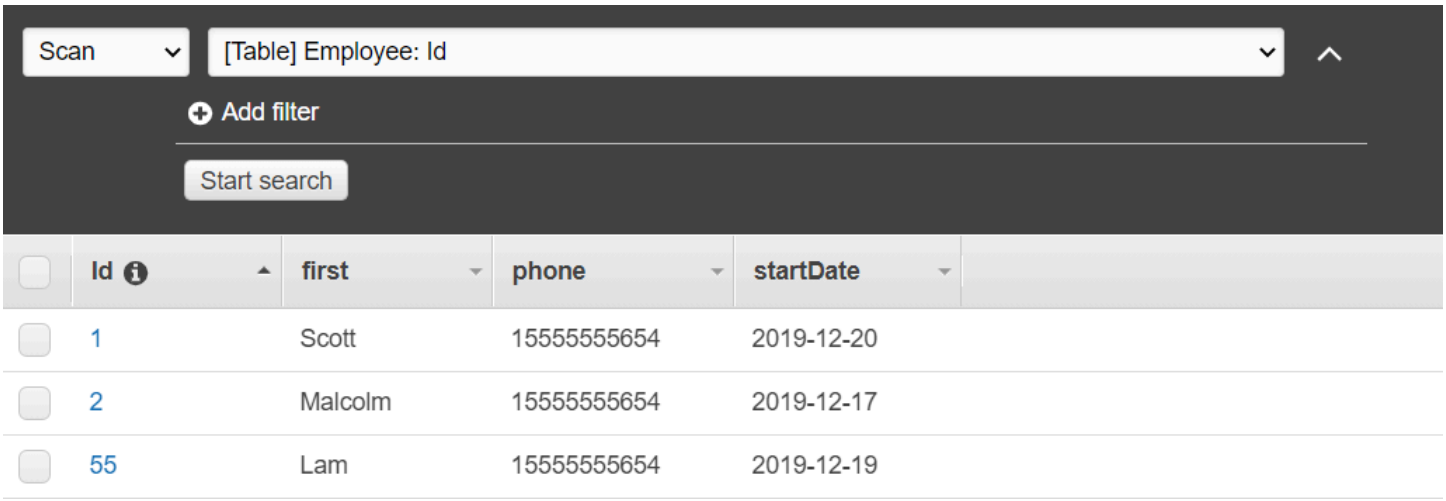
このチュートリアルは完了までに約 20 分かかります。

このチュートリアルでは、JavaScript ロジックを使用して、このユースケースを実行するソリューションを作成する方法を示します。例えば、データベースを読み取り、1 年記念日に達した従業員を特定する方法、データを処理する方法、Lambda 関数を使用してテキストメッセージを送信する方法について説明します。次に、cron 式を使用して Lambda 関数を毎日平日に呼び出す方法を説明します。

この AWS チュートリアルでは、これらのフィールドを含む Employee という名前の Amazon DynamoDB テーブルを使用します。

- id - 表のプライマリキー。

- 名前 - 従業員のファーストネーム。
- 電話 - 従業員の電話番号。
- 開始日 - 従業員の入社日。



> **Important**
>
> 完了コスト: このドキュメントに含まれる AWS サービスは、 AWS 無料利用枠に含まれています。ただし、このチュートリアルを完了した後は、必ずすべてのリソースを終了して料金が発生しないようにしてください。

アプリケーションを構築するには、

1. 前提条件を満たします
2. AWS リソースを作成する
3. ブラウザスクリプトを準備
4. Lambda 関数の作成とアップロード
5. Lambda 関数をデプロイします
6. アプリを実行
7. リソースを削除します

前提条件タスク

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node.js TypeScript の例を実行するようにプロジェクト環境を設定し、必要な AWS SDK for JavaScript およびサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

AWS リソースを作成する

このチュートリアルでは、以下のリソースが必要です。

- Idという名前のキーと前の図に示されているフィールドを持つEmployeeという「Amazon Dyn[oDB」(Amazon DynamoDB)の表。このユースケースでテストする有効な携帯番号を含め、正しいデータを入力してください。詳細については、テーブルの作成を参照してください。
- Lambda関数を実行するためのアクセス許可が付与されたIAMロール。
- Lambda 関数をホストするAmazon S3 バケット。

これらのリソースは手動で作成できますが、このチュートリアルで説明されている AWS CloudFormation ように、 を使用してこれらのリソースをプロビジョニングすることをお勧めします。

を使用して AWS リソースを作成する AWS CloudFormation

AWS CloudFormation を使用すると、 AWS インフラストラクチャのデプロイを予測どおりに繰り返し作成およびプロビジョニングできます。詳細については AWS CloudFormation、AWS CloudFormation 「 ユーザーガイド」を参照してください。

を使用して AWS CloudFormation スタックを作成するには AWS CLI :

1. AWS CLI 「 ユーザーガイド」の手順に従って AWS CLI 、 をインストールして設定します。
2. プロジェクトフォルダのルートディレクトリで、`setup.yaml` という名前のファイルを作成し、それにこの GitHub にコンテンツをコピーします。

> **Note**
>
> AWS CloudFormation テンプレートは、 AWS CDK GitHub で利用可能な を使用して生成されました。の詳細については AWS CDK、「 AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」を参照してください。

3. コマンドラインから以下のコマンドを実行し、「*STACK_NAME*」をスタックの一意の名前に置き換えます。

> **Important**
>
> スタック名は、 AWS リージョンと AWS アカウント内で一意である必要があります。最大 128 文字まで指定でき、数字とハイフンを使用できます。

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

`create-stack` コマンドパラメータの詳細については、 AWS CLI Command Reference guide (コマンドリファレンスガイド) および「AWS CloudFormation ユーザーガイド」を参照してください。

AWS CloudFormation ダッシュボードでスタックを開き、リソースタブを選択して、コンソールでリソースのリストを表示します。チュートリアルにはこれらが必要です。

4. スタックが作成されたら、 AWS SDK for JavaScript 「」で説明されているように、 を使用して DynamoDB テーブルにデータを入力しますDynamoDB 表にデータを入力します。。

DynamoDB 表にデータを入力します。

テーブルにデータを入力するには、まず `libs` という名前のディレクトリを作成し、そこに `dynamoClient.js` という名前のファイルを作成し、それに以下の内容を貼り付けます。

```
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 };
```

このコードはこのGitHubで利用できます。

次に、populate-table.js というファイルをプロジェクトフォルダのルートディレクトリに作成し、この GitHub にコンテンツをコピーします。項目の1つについて、phone のプロパティの値をE.164形式の有効な携帯電話番号に置き換え、startDate の値を今日の日付に置き換えます。

コマンドラインから、以下のコマンドを実行します。

```
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();
```

このコードはこのGitHubで利用できます。

AWS Lambda 関数の作成

SDK の設定

まず、必要な AWS SDK for JavaScript (v3) モジュールとコマンドをインポートします。DynamoDBClient と DynamoDB ScanCommand、、SNSClientおよび Amazon SNS PublishCommand コマンド。*REGION* を AWS リージョンに置き換えます。次に、今日の日付を計算し、パラメータに割り当てます。次に、ScanCommandパラメータを作成します。*TABLE_NAME*を、この例のAWS リソースを作成する セクションで作成したテーブルの名に置き換えます。

以下のコードスニペットは、このステップを示しています (詳細な例については、Lambda 関数をバンドルします を参照してください)。

```
"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",
};
```

DynamoDB テーブルをスキャンします

まず、Amazon SNS PublishCommand を使用してテキストメッセージを公開するために sendText と呼ばれる非同期/待機関数を作成します。次に、今日が勤務記念日である従業員の DynamoDB テーブルをスキャンし、sendText 関数を呼び出してこれらの従業員にテキストメッセージを送信する try ブロックパターンを追加します。エラーが発生した場合は、catch ブロックされます。

以下のコードスニペットは、このステップを示しています (詳細な例については、Lambda 関数をバンドルします を参照してください)。

```
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);
  }
};
```

Lambda 関数をバンドルします

このトピックでは、 mylambdafunction.jsと、この例に必要な AWS SDK for JavaScript モジュールを というバンドルされたファイルにバンドルする方法について説明しますindex.js。

1. まだの場合は、この例の前提条件タスクに従ってwebpackをインストールしてください。

> **Note**
>
> Webpack の詳細については、アプリケーションを webpack にバンドルする を参照してください。

2. コマンドラインで以下を実行して、この例のJavaScriptを<index.js>というファイルにバンドルします。

```
webpack mylamdbafunction.js --mode development --target node --devtool false --
output-library-target umd -o index.js
```

> **Important**
>
> 出力の名前がindex.jsであることに注意してください。Lambda関数が機能するにはindex.jsハンドラーが必要です。

3. バンドルされた出力ファイル index.js を、my-lambda-function.zipという名前の ZIP ファイルに圧縮します。
4. このチュートリアルのAWS リソースを作成する トピックで作成したAmazonS3バケットにmylambdafunction.zipをアップロードします。

これはmylambdafunction.jsの完全なブラウザスクリプトコードです。

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

Lambda 関数をデプロイします

プロジェクトのルートで、`lambda-function-setup.js` ファイルを作成し、それに以下の内容をペーストします。

BUCKET_NAME を Lambda 関数の ZIP バージョンをアップロードした Amazon S3 バケットの名前に置き換えます。*ZIP_FILE_NAME*（ZIPファイル名）を、Lambda関数のZIPバージョンの名前に置き換えます。*IAM_ROLE_ARN*を、このチュートリアルのAWS リソースを作成する のトピックで作成したIAMロールのAmazonリソース番号（ARN）に置き換えま

す。*LAMBDA_FUNCTION_NAME*（Lambdaファンクション名）をLambda関数の名前に置き換えます。

```
// 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();
```

コマンドラインで次を入力して、Lambda 関数をデプロイします。

```
node lambda-function-setup.js
```

このコード例はこのGitHubに で利用可能です。

Lambda 関数を呼び出す CloudWatch を設定します

Lambda 関数を呼び出す CloudWatch を設定するには、

1. Lambda コンソールで [Functions (関数)] ページを開きます。
2. Lambda 関数を選択します。
3. [Designer] で、[Add trigger] を選択します。
4. トリガーの種類をCloudWatch Events/EventBridgeに設定します。
5. ルールで、 Create a new rule (新規ルールの作成) を選択します。
6. ルール名とルールの説明を入力します。
7. ルールタイプで、 Schedule expression (スケジュール式) を選びます。
8. Schedule expression (スケジュール式) フィールドには、cron 式を入力します。例えば、cron(0 12 ? * MON-FRI *) (cron (0 12? * 月-金 *)) 。
9. [追加] を選択します。

> **Note**
>
> 詳細については、「 Using Lambda with CloudWatch Events」 (CloudWatch イベントでLambda を使用) を参照してください。

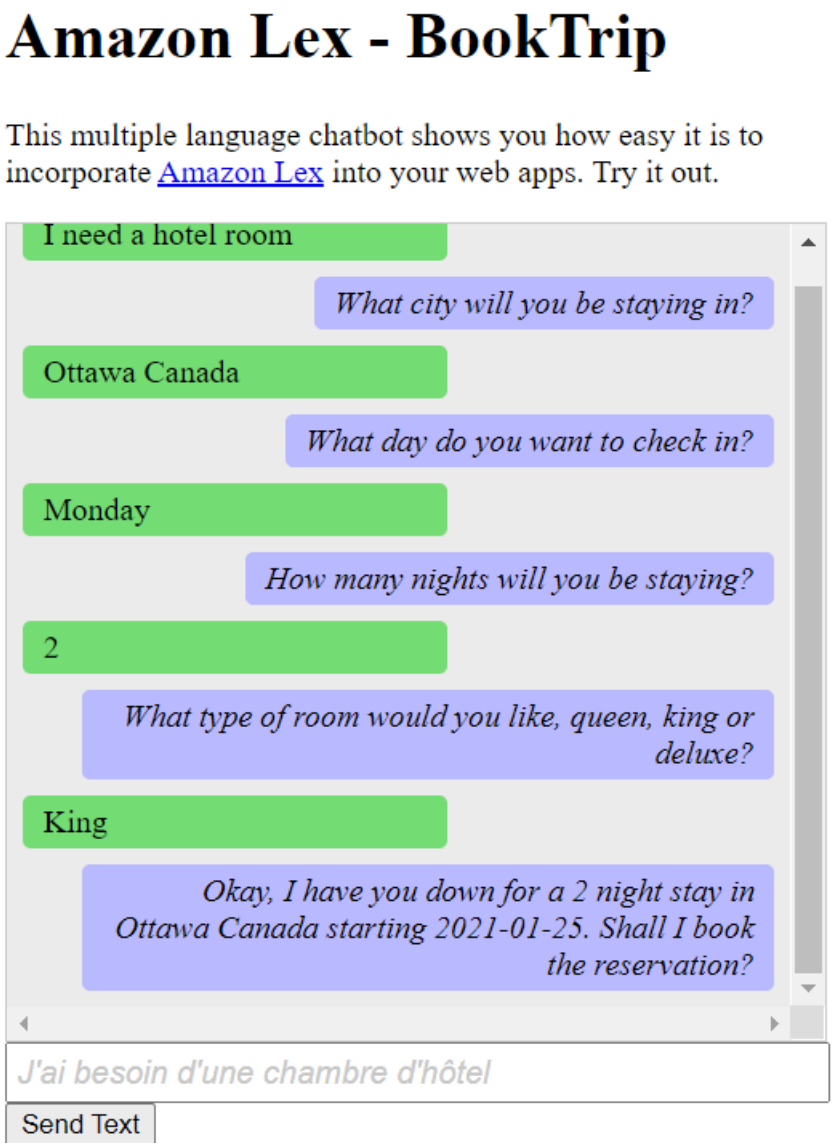
リソースを削除します

お疲れ様でした。 AWS SDK for JavaScriptを使用してAmazon CloudWatch スケジュールイベントを通じて Lambda 関数を呼び出します。このチュートリアルの冒頭で説明したように、このチュートリアルを進めたうえで、作成したすべてのリソースを終了して、料金が発生しないようにしてください。これを行うには、このチュートリアルの AWS リソースを作成する トピックで作成した AWS CloudFormation スタックを次のように削除します。

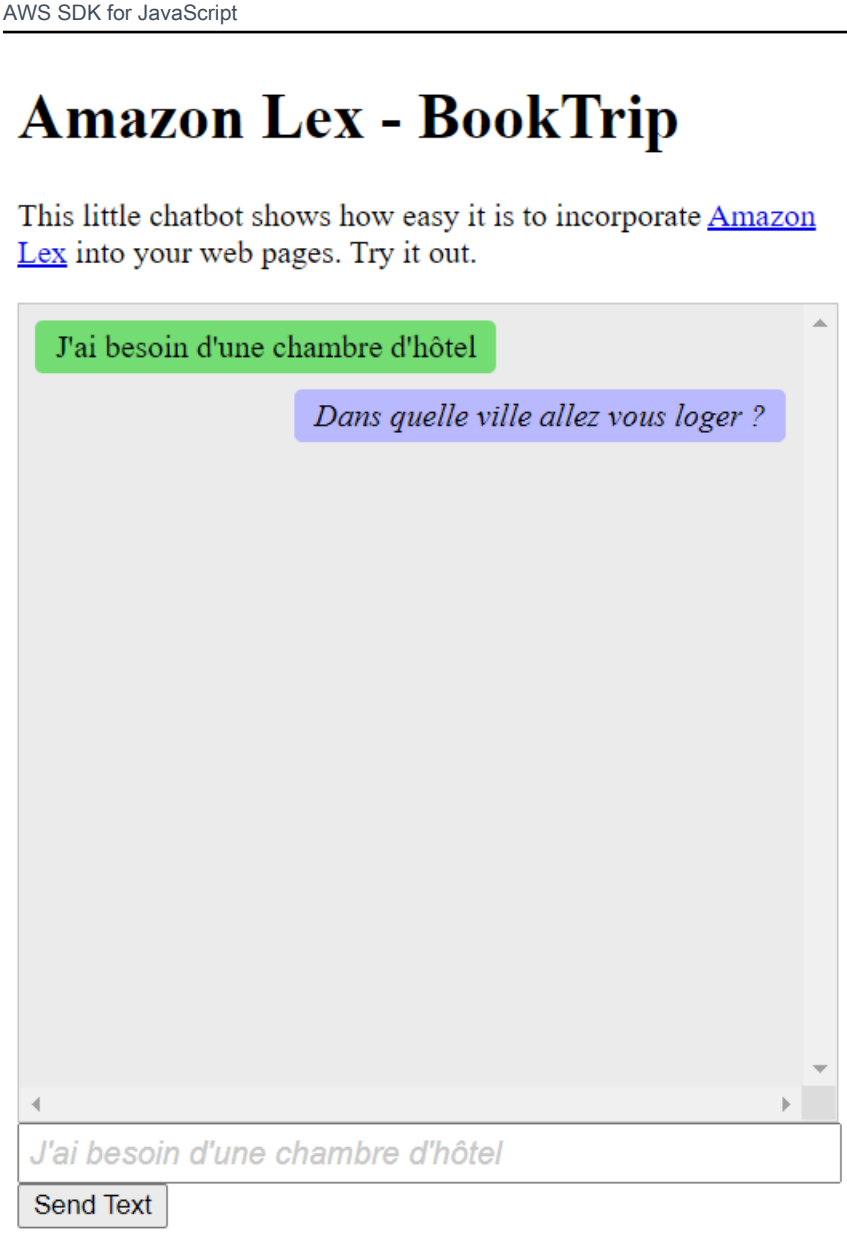
1. AWS CloudFormation コンソールを開きます。
2. 「スタック」ページで、スタックを選択します。
3. [削除] を選択します。

Amazon Lex chatbotを構築する

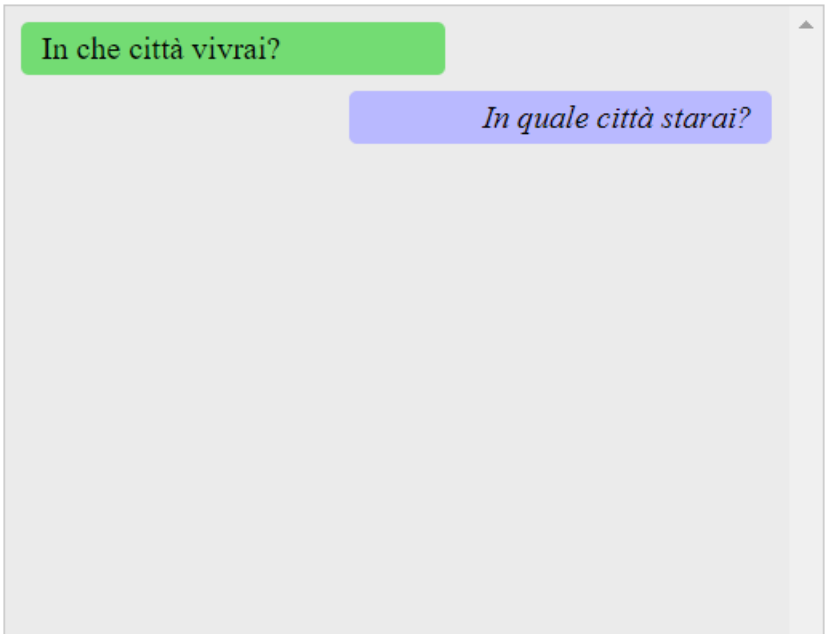
ウェブアプリケーション内に Amazon Lex chatbotを作成して、ウェブサイトの訪問者に対応することができます。Amazon Lex chatbotは、人と直接連絡することなく、ユーザーとのオンラインチャットの会話を実行する機能です。例えば、次の図は、ホテルの部屋の予約についてユーザーに対応する Amazon Lex chatbotを示しています。



この AWS チュートリアルで作成した Amazon Lex チャットボットは、複数の言語を処理できます。例えば、フランス語を話すユーザーは、フランス語のテキストを入力し、フランス語で応答を返すことができます。



同様に、ユーザーはイタリア語でAmazon Lex chatbotで通信できます。



この AWS チュートリアルでは、Amazon Lex チャットボットを作成し、Node.js ウェブアプリケーションに統合する方法について説明します。 AWS SDK for JavaScript (v3) は、以下の AWS サービスを呼び出すために使用されます。

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

完了コスト： このドキュメントに含まれる AWS サービスは、 AWS 無料利用枠に含まれています。

注意: このチュートリアルを進めるうえで作成したすべてのリソースを終了して、料金が発生しないようにしてください。

アプリを構築するには

1. 前提条件
2. リソースのプロビジョニング
3. Amazon Lex chatbotの作成する

4. HTMLを作成する
5. ブラウザスクリプトを作成
6. 次のステップ

前提条件

この例をセットアップして実行するには、まず次のタスクを完了する必要があります。

- これらの Node TypeScript の例を実行するようにプロジェクト環境をセットアップし、必要な AWS SDK for JavaScript モジュールとサードパーティーモジュールをインストールします。「GitHub」の指示に従います。
- ユーザーの認証情報を使用して、共有設定ファイルを作成します。共有認証情報ファイルの提供の詳細については、「AWS SDK とツールのリファレンスガイド」の「共有設定ファイルおよび認証情報ファイル」を参照してください。

> **Important**
>
> この例では、ECMAScript6 (ES6) を使用しています。これには Node.js バージョン13.x 以降が必要です。Node.js の最新バージョンをダウンロードしてインストールするには、「Node.js ダウンロード」を参照してください。
> ただし、CommonJS 構文を使用したい場合は、「JavaScript ES6/CommonJS 構文」を参照してください。

AWS リソースを作成する

このチュートリアルでは、以下のリソースが必要です。

- 次の権限がアタッチされた非認証IAM ロールです。
  - Amazon Comprehend
  - Amazon Translate
  - Amazon Lex

このリソースは手動で作成できますが、このチュートリアルで説明されている AWS CloudFormation ように、 を使用してこれらのリソースをプロビジョニングすることをお勧めします。

を使用して AWS リソースを作成する AWS CloudFormation

AWS CloudFormation を使用すると、 AWS インフラストラクチャのデプロイを予測どおりに繰り返し作成およびプロビジョニングできます。詳細については AWS CloudFormation、AWS CloudFormation 「 ユーザーガイド」を参照してください。

を使用して AWS CloudFormation スタックを作成するには AWS CLI :

1. AWS CLI 「 ユーザーガイド AWS CLI 」の手順に従って、 をインストールして設定します。
2. プロジェクトフォルダのルートディレクトリで、`setup.yaml` という名前のファイルを作成し、それにこの GitHub にコンテンツをコピーします。

   > **Note**
   >
   > AWS CloudFormation テンプレートは、 AWS CDK GitHub で利用可能な を使用して生成されました。の詳細については AWS CDK、「 AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」を参照してください。

3. コマンドラインから以下のコマンドを実行し、「*`STACK_NAME`*」をスタックの一意の名前に置き換えます。

   > **Important**
   >
   > スタック名は、 AWS リージョンと AWS アカウント内で一意である必要があります。最大 128 文字まで指定でき、数字とハイフンを使用できます。

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

   `create-stack` コマンドパラメータの詳細については、 AWS CLI Command Reference guide (コマンドリファレンスガイド) および「AWS CloudFormation ユーザーガイド」を参照してください。

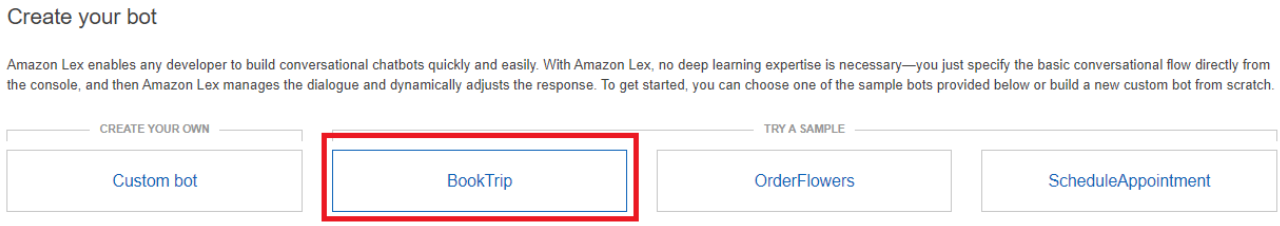
   作成されたリソースを表示するには、Amazon Lex コンソールを開き、スタックを選択し、Resources (リソース)タブを選びます。

Amazon Lex botを作成します

> **Important**
>
> Amazon Lex V1 コンソールを使用してボットを作成します。この例は V2 を使用して作成されたボットでは機能しません。

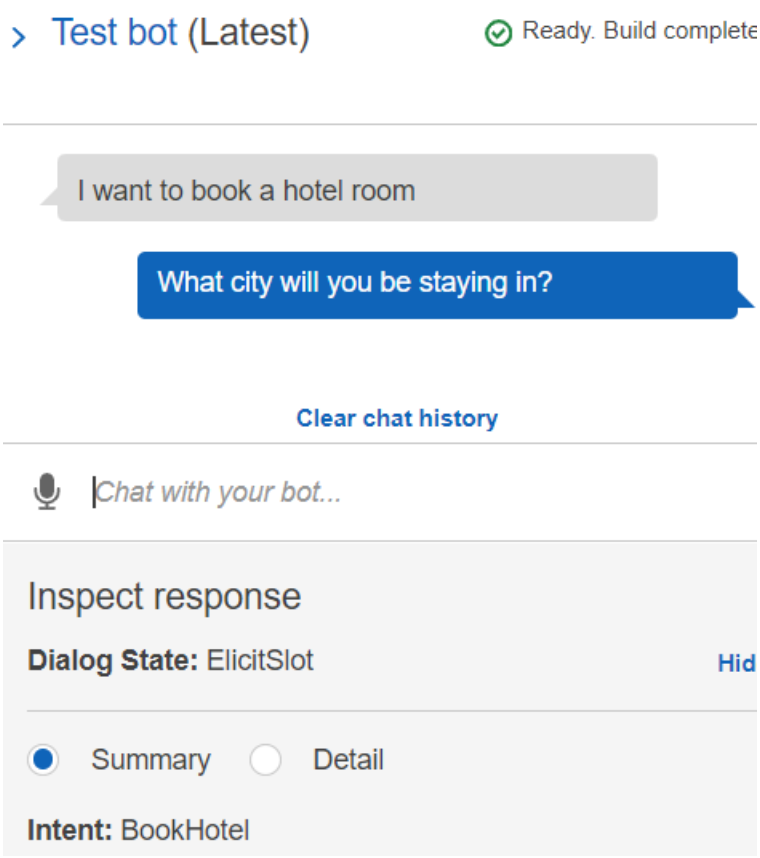
最初のステップは、Amazon Web Services マネジメントコンソールを使用して Amazon Lex chatbot を作成することです。この例では、Amazon LexBookTripの例が使用されます。詳細については「Book Trip (出張の予約)」を参照してください。

- Amazon Web Services マネジメントコンソールにサインインして、Amazon Web Services Console (Amazon Web Services コンソール)のAmazon Lex コンソールを開きます。
- Botsページで、Create (作成) を選択します。
- BookTripブループリント (デフォルトのボット名はBookTripのままにしておきます) を選択します。



- デフォルト設定を入力し、作成(コンソールは BookTripボットを表示します) を選択します。編集タブで、事前設定されているインテントの詳細を確認します。

- テストウィンドウでボットをテストします。ホテルの部屋を予約したいと入力してテストを開始します。



- 発行を選択し、エイリアス名を指定します (を使用する場合はこの値が必要です AWS SDK for JavaScript) 。

> **Note**
>
> JavaScript コードの bot name (ボット名) と bot alias (ボットエイリアス) を参照する必要があります。

HTMLを作成する

`index.html` という名前のファイルを作成します。以下のコードをコピーして、`index.html`に貼り付けます。このHTMLは`main.js`を参照します。これは index.js のバンドルバージョンで、必要な AWS SDK for JavaScript モジュールが含まれています。このファイルは HTMLを作成する で作成します。`index.html` はスタイルを追加する `style.css` も参照します。

```
<!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>
```

このコードもこのGitHubで公開されています。

ブラウザスクリプトを作成する

index.js という名前のファイルを作成します。以下のコードをコピーして、index.jsに貼り付けます。必要な AWS SDK for JavaScript モジュールとコマンドをインポートします。Amazon Lex、Amazon Comprehend、およびAmazon Translate のクライアントを作成します。*REGION* を AWS Region に、*IDENTITY_POOL_ID* を で作成した ID プールの ID に置き換えますAWS リソースを作成する 。このアイデンティティプールのIDを取得するには、Amazon Cognito コンソールでアイデンティティプールを開き、Edit identity pool (アイデンティティプールの編集) を選択し、サイドメ

ニューのSample code (サンプルコード)を選択します。アイデンティティプールの IDはコンソールに赤いテキストで表示されます。

まず、libsディレクトリを作成し、3つのファイル、comprehendClient.js、lexClient.js、translateClient.jsを作成することで、必要なサービスクライアントオブジェクトを作成します。以下の適切なコードをそれぞれに貼り付け、*REGION* および *IDENTITY_POOL_ID* を各ファイルで置き換えます。

> **Note**
>
> を使用して AWS リソースを作成する AWS CloudFormationで作成したAmazon CognitoアイデンティティプールのIDを使用します。

```
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 };
```

このコードは このGitHubに で利用可能です。

次に、index.jsファイルを作成し、そこへ以下のコードを貼り付けます。

*BOT_ALIAS*と*BOT_NAME*をAmazon Lex ボットのエイリアスと名前にそれぞれ置き換え、*USER_ID*をユーザー ID に置き換えます。createResponse非同期関数は以下を実行します。

- ユーザーが入力したテキストをブラウザに取り込み、Amazon Comprehend を使用して言語コードを決定します。
- 言語コードを取得し、Amazon Translate を使用してテキストを英語に翻訳します。
- 翻訳されたテキストを取得し、Amazon Lex を使用してレスポンスを生成します。

- レスポンスをブラウザページに投稿します。

```
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;
```

このコードは このGitHubに で利用可能です。

ここで、webpack を使用して `index.js`および AWS SDK for JavaScript モジュールを 1 つのファイルにバンドルします`main.js`。

1. まだの場合は、この例の前提条件に従ってwebpackをインストールしてください。

   > **Note**
   >
   > Webpack の詳細については、アプリケーションを webpack にバンドルする を参照してください。

2. コマンドラインで以下を実行して、この例の JavaScriptを`main.js`というファイルにバンドルします。

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

次のステップ

お疲れ様でした。Amazon Lex を使用してインタラクティブなユーザーエクスペリエンスを実現する Node.js アプリケーションが作成されました。このチュートリアルの冒頭で説明したように、チュートリアルを進めるうえで作成したすべてのリソースを終了して、料金が発生しないようにしてください。これを行うには、このチュートリアルの AWS リソースを作成する トピックで作成した AWS CloudFormation スタックを次のように削除します。

1. AWS CloudFormation コンソールを開きます。
2. 「スタック」ページで、スタックを選択します。
3. [削除] を選択します。

AWS クロスサービスの例については、AWS SDK for JavaScript 「クロスサービスの例」を参照してください。

SDK for JavaScript (v3) のコード例

このトピックのコード例は、 で AWS SDK for JavaScript (v3) を使用する方法を示しています AWS。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

一部のサービスには、サービス固有のライブラリや関数の活用方法を示す追加のカテゴリ例が含まれています。

サービス

- SDK for JavaScript (v3) を使用した API Gateway の例
- SDK for JavaScript (v3) を使用した Aurora の例
- SDK for JavaScript (v3) を使用した自動スケーリングの例
- SDK for JavaScript (v3) を使用した Amazon Bedrock の例
- SDK for JavaScript (v3) を使用する Amazon Bedrock ランタイムの例
- SDK for JavaScript (v3) を使用した Amazon Bedrock エージェントの例
- SDK for JavaScript (v3) を使用した Amazon Bedrock エージェントランタイムの例
- SDK for JavaScript (v3) を使用した CloudWatch の例
- SDK for JavaScript (v3) を使用した CloudWatch Events の例
- SDK for JavaScript (v3) を使用した CloudWatch Logs の例
- SDK for JavaScript (v3) を使用した CodeBuild の例
- SDK for JavaScript (v3) を使用した Amazon Cognito ID の例
- SDK for JavaScript (v3) を使用した Amazon Cognito ID プロバイダーの例
- SDK for JavaScript (v3) を使用した Amazon Comprehend の例
- SDK for JavaScript (v3) を使用した Amazon DocumentDB の例
- SDK for JavaScript (v3) を使用した DynamoDB の例
- SDK for JavaScript (v3) を使用した Amazon EC2 の例

- Elastic Load Balancing - SDK for JavaScript (v3) を使用したバージョン 2 の例
- AWS Entity Resolution SDK for JavaScript (v3) を使用した の例
- SDK for JavaScript (v3) を使用した EventBridge の例
- AWS Glue SDK for JavaScript (v3) を使用した例
- SDK for JavaScript (v3) を使用した HealthImaging の例
- SDK for JavaScript (v3) を使用した IAM の例
- AWS IoT SiteWise SDK for JavaScript (v3) を使用した の例
- SDK for JavaScript (v3) を使用した Kinesis の例
- SDK for JavaScript (v3) を使用した Lambda 例
- SDK for JavaScript (v3) を使用した Amazon Lex の例
- SDK for JavaScript (v3) を使用した Amazon Location の例
- SDK for JavaScript (v3) を使用した Amazon MSK の例
- SDK for JavaScript (v3) を使用した Amazon Personalize の例
- SDK for JavaScript (v3) を使用した Amazon Personalize Events の例
- SDK for JavaScript (v3) による Amazon Personalize Runtime の例
- SDK for JavaScript (v3) を使用した Amazon Pinpoint の例
- SDK for JavaScript (v3) を使用した Amazon Polly の例
- SDK for JavaScript (v3) を使用した Amazon RDS の例
- SDK for JavaScript (v3) を使用した Amazon RDS Data Service の例
- SDK for JavaScript (v3) を使用した Amazon Redshift の例
- SDK for JavaScript (v3) を使用した Amazon Rekognition の例 JavaScript
- SDK for JavaScript (v3) を使用した Amazon S3 の例
- SDK for JavaScript (v3) を使用した S3 Glacier の例
- SDK for JavaScript (v3) を使用した SageMaker AI の例
- SDK for JavaScript (v3) を使用した Secrets Manager の例
- SDK for JavaScript (v3) を使用した Amazon SES の例
- SDK for JavaScript (v3) を使用した Amazon SNS の例
- SDK for JavaScript (v3) を使用した Amazon SQS の例
- SDK for JavaScript (v3) を使用した Step Functions の例
- AWS STS SDK for JavaScript (v3) を使用した例

- サポート SDK for JavaScript (v3) を使用した例
- SDK for JavaScript (v3) を使用した Systems Manager の例
- SDK for JavaScript (v3) を使用した Amazon Textract の例
- SDK for JavaScript (v3) を使用した Amazon Transcribe の例
- SDK for JavaScript (v3) を使用した Amazon Translate の例

SDK for JavaScript (v3) を使用した API Gateway の例

次のコード例は、API Gateway で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- シナリオ

シナリオ

サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway

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

API Gateway を使用して Lambda 関数を呼び出す

次のコード例は、Amazon API Gateway によって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

Lambda JavaScript ランタイム API を使用して AWS Lambda 関数を作成する方法を示します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、Amazon API Gateway によって呼び出される Lambda 関数を作成する方法を示します。この関数は、Amazon DynamoDB テーブルをスキャンして、Amazon Simple Notification Service (Amazon SNS) を使用して、従業員に年間の記念日を祝福するテキストメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

SDK for JavaScript (v3) を使用した Aurora の例

次のコード例は、Aurora で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

シナリオ

Aurora Serverless 作業項目トラッカーの作成

次のコード例は、Amazon Aurora Serverless データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを送信するウェブアプリケーションを作成する方法を示しています。

SDK for JavaScript (v3)

AWS SDK for JavaScript (v3) を使用して Amazon Aurora データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信するウェブアプリケーションを作成する方法を示します。この例では、React.js で構築されたフロントエンドを使用して Express Node.js バックエンドと対話します。

- React.js ウェブアプリケーションを と統合します AWS のサービス。
- Aurora テーブルの項目を一覧表示、追加、更新します。
- Amazon SES を使用して、フィルター処理された作業項目の E メールレポートを送信します。
- 含まれている AWS CloudFormation スクリプトを使用してサンプルリソースをデプロイおよび管理します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス

- Amazon SES

SDK for JavaScript (v3) を使用した自動スケーリングの例

次のコード例は、Auto Scaling で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

アクション

`AttachLoadBalancerTargetGroups`

次の例は、`AttachLoadBalancerTargetGroups` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const client = new AutoScalingClient({});
await client.send(
```

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AttachLoadBalancerTargetGroups」を参照してください。

シナリオ

レジリエントなサービスの構築と管理

次のコード例は、本、映画、曲のレコメンデーションを返す負荷分散型ウェブサービスの作成方法を示しています。この例は、障害に対するサービスの対応方法と、障害発生時の耐障害性を高めるためにサービスを再構築する方法を示しています。

- Amazon EC2 Auto Scaling グループを使用して、起動テンプレートに基づいて Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを作成し、インスタンス数を所定の範囲内に維持します。
- Elastic Load Balancing で HTTP リクエストを処理して配信します。
- Auto Scaling グループ内のインスタンスの状態を監視し、正常なインスタンスにのみリクエストを転送します。
- 各 EC2 インスタンスで Python ウェブサーバーを実行して HTTP リクエストを処理します。ウェブサーバーはレコメンデーションとヘルスチェックを返します。
- Amazon DynamoDB テーブルを使用してレコメンデーションサービスをシミュレートできます。
- AWS Systems Manager パラメータを更新して、リクエストとヘルスチェックに対するウェブサーバーの応答を制御します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
#!/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.",
  });
}
```

すべてのリソースをデプロイするための手順を作成します。

```
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,
];
```

デモを実行するための手順を作成します。

```
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;
}
```

すべてのリソースを破棄するための手順を作成します。

```
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.`);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - 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

SDK for JavaScript (v3) を使用した Amazon Bedrock の例

次のコード例は、Amazon Bedrock で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Amazon Bedrock

次のコード例は、Amazon Bedrock の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListFoundationModels」を参照してください。

トピック

- アクション

アクション

`GetFoundationModel`

次の例は、`GetFoundationModel` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

基盤モデルに関する詳細を取得します。

```
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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetFoundationModel」を参照してください。

`ListFoundationModels`

次の例は、ListFoundationModels を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

使用可能な基盤モデルを一覧表示します。

```
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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListFoundationModels」を参照してください。

SDK for JavaScript (v3) を使用する Amazon Bedrock ランタイムの例

次のコード例は、Amazon Bedrock ランタイムで AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Amazon Bedrock

次のコード例は、Amazon Bedrock の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/**
 * @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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InvokeModel」を参照してください。

トピック

- シナリオ
- Amazon Nova
- Amazon Nova Canvas
- Amazon Titan Text
- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

シナリオ

Amazon Bedrock で複数の基盤モデルを呼び出す

次のコード例は、Amazon Bedrock のさまざまな大規模言語モデル (LLMs) にプロンプトを準備して送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - InvokeModel
  - InvokeModelWithResponseStream

Converse API でのツールの使用

次のコード例は、アプリケーション、生成 AI モデル、接続されたツールまたは API 間の一般的なインタラクションを構築し、AI と外部世界のインタラクションを仲介する方法を示しています。外部気象 API を AI モデルに接続する例を使用して、ユーザー入力に基づいてリアルタイムの気象情報を提供します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

シナリオフローのプライマリ実行。このシナリオはユーザー、Amazon Bedrock Converse API、気象ツールの間の会話を調整します。

```
/* 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 { dirname } from "node:path";
const __filename = fileURLToPath(import.meta.url);
import data from "./questions.json" with { type: "json" };
import toolConfig from "./tool_config.json" with { type: "json" };

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",
});

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 });
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Converse」を参照してください。

Amazon Nova

Converse

次のコード例は、Bedrock の Converse API を使用して Amazon Nova にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Amazon Nova にテキストメッセージを送信します。

```
// 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;
}
```

ツール設定を含む Bedrock の Converse API を使用して Amazon Nova にメッセージの会話を送信します。

```
// 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);
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Amazon Nova にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Amazon Nova にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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;
}
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「ConverseStream」を参照してください。

シナリオ: Converse API でのツールの使用

次のコード例は、アプリケーション、生成 AI モデル、接続されたツールまたは API 間の一般的なインタラクションを構築し、AI と外部世界のインタラクションを仲介する方法を示しています。外部気象 API を AI モデルに接続する例を使用して、ユーザー入力に基づいてリアルタイムの気象情報を提供します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

シナリオフローのプライマリ実行。このシナリオはユーザー、Amazon Bedrock Converse API、気象ツールの間の会話を調整します。

```
/* 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 { dirname } from "node:path";
const __filename = fileURLToPath(import.meta.url);
import data from "./questions.json" with { type: "json" };
import toolConfig from "./tool_config.json" with { type: "json" };

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",
});

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 });
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Converse」を参照してください。

Amazon Nova Canvas

InvokeModel

次のコード例は、Amazon Bedrock で Amazon Nova Canvas を呼び出してイメージを生成する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon Nova Canvas で画像を作成します。

```
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;
    });
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InvokeModel」を参照してください。

Amazon Titan Text

Converse

次のコード例は、Bedrock の Converse API を使用して Amazon Titan Text にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Amazon Titan Text にテキストメッセージを送信します。

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

import {
  BedrockRuntimeClient,
  ConverseCommand,
```

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

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

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

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

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

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

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Amazon Titan Text にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Amazon Titan Text にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

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

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

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

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

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

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

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

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

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

- API の詳細については、「AWS SDK for JavaScript API Reference」の「ConverseStream」を参照してください。

InvokeModel

次のコード例は、Invoke Model API を使用して Amazon Titan Text にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Invoke Model API を使用してテキストメッセージを送信します。

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

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

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

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

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

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

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

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

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InvokeModel」を参照してください。

Anthropic Claude

Converse

次のコード例は、Bedrock の Converse API を使用して Anthropic Claude にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して、Anthropic Claude にテキストメッセージを送信します。

```
// 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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Anthropic Claude にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Anthropic Claude にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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);
}
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「ConverseStream」を参照してください。

InvokeModel

次のコード例は、Invoke Model API を使用して Anthropic Claude にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Invoke Model API を使用してテキストメッセージを送信します。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InvokeModel」を参照してください。

InvokeModelWithResponseStream

次のコード例は、Invoke Model API を使用して Anthropic Claude モデルにテキストメッセージを送信し、レスポンスストリームを印刷する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Invoke Model API を使用してテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

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

- API の詳細については、AWS SDK for JavaScript API リファレンス」の「InvokeModelWithResponseStream」を参照してください。

Cohere Command

Converse

次のコード例は、Bedrock の Converse API を使用して Cohere Command にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Cohere Command にテキストメッセージを送信します。

```
// 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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Cohere Command にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Cohere Command にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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);
}
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「ConverseStream」を参照してください。

Meta Llama

Converse

次のコード例は、Bedrock の Converse API を使用して Meta Llama にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Meta Llama にテキストメッセージを送信します。

```
// 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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Meta Llama にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Meta Llama にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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);
}
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「ConverseStream」を参照してください。

InvokeModel

次のコード例は、Invoke Model API を使用して Meta Llama にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Invoke Model API を使用してテキストメッセージを送信します。

```
// 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
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InvokeModel」を参照してください。

InvokeModelWithResponseStream

次のコード例は、Invoke Model API を使用して Meta Llama にテキストメッセージを送信し、レスポンスストリームを印刷する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Invoke Model API を使用してテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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
```

- API の詳細については、AWS SDK for JavaScript API リファレンス」の「InvokeModelWithResponseStream」を参照してください。

Mistral AI

Converse

次のコード例は、Bedrock の Converse API を使用して Mistral にテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Mistral にテキストメッセージを送信します。

```
// 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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Mistral にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Mistral にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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);
}
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「ConverseStream」を参照してください。

InvokeModel

次のコード例は、Invoke Model API を使用して Mistral モデルにテキストメッセージを送信する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Invoke Model API を使用してテキストメッセージを送信します。

```
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);
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InvokeModel」を参照してください。

SDK for JavaScript (v3) を使用した Amazon Bedrock エージェントの例

次のコード例は、Amazon Bedrock エージェントで AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Amazon Bedrock エージェントの始め方

次のコード例は、Amazon Bedrock エージェントの使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - GetAgent
  - ListAgents

トピック

- アクション

アクション

`CreateAgent`

次の例は、`CreateAgent` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

エージェントを作成します。

```
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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateAgent」を参照してください。

`DeleteAgent`

次の例は、`DeleteAgent` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

エージェントを削除します。

```
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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteAgent」を参照してください。

GetAgent

次の例は、GetAgent を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

エージェントを取得します。

```
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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetAgent」を参照してください。

`ListAgentActionGroups`

次の例は、`ListAgentActionGroups` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

エージェントのアクショングループを一覧表示します。

```
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);
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListAgentActionGroups」を参照してください。

ListAgents

次の例は、ListAgents を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アカウントに属するエージェントを一覧表示します。

```
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);
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListAgents」を参照してください。

SDK for JavaScript (v3) を使用した Amazon Bedrock エージェントランタイムの例

次のコード例は、Amazon Bedrock エージェントランタイムで AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`InvokeAgent`

次の例は、InvokeAgent を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InvokeAgent」を参照してください。

InvokeFlow

次の例は、InvokeFlow を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InvokeFlow」を参照してください。

SDK for JavaScript (v3) を使用した CloudWatch の例

次のコード例は、CloudWatch で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`DeleteAlarms`

次の例は、`DeleteAlarms` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteAlarms」を参照してください。

`DescribeAlarmsForMetric`

次の例は、`DescribeAlarmsForMetric` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
```

```
export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeAlarmsForMetric」を参照してください。

`DisableAlarmActions`

次の例は、`DisableAlarmActions` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API 詳細については、「AWS SDK for JavaScript API リファレンス」の「DisableAlarmActions」を参照してください。

`EnableAlarmActions`

次の例は、`EnableAlarmActions` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「EnableAlarmActions」を参照してください。

`ListMetrics`

次の例は、`ListMetrics` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
    }
  }
};
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListMetrics」を参照してください。

PutMetricAlarm

次の例は、PutMetricAlarm を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutMetricAlarm」を参照してください。

`PutMetricData`

次の例は、`PutMetricData` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutMetricData」を参照してください。

SDK for JavaScript (v3) を使用した CloudWatch Events の例

次のコード例は、CloudWatch Events で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`PutEvents`

次の例は、`PutEvents` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「PutEvents」を参照してください。

`PutRule`

次の例は、`PutRule` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「PutRule」を参照してください。

PutTargets

次の例は、PutTargets を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";
```

```
export const client = new CloudWatchEventsClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「PutTargets」を参照してください。

SDK for JavaScript (v3) を使用した CloudWatch Logs の例

次のコード例は、CloudWatch Logs で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

アクション

`CreateLogGroup`

次の例は、`CreateLogGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateLogGroup」を参照してください。

`DeleteLogGroup`

次の例は、`DeleteLogGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteLogGroup」を参照してください。

`DeleteSubscriptionFilter`

次の例は、`DeleteSubscriptionFilter` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
```

- API の詳細については、「AWS SDK for JavaScript SDK for Kotlin API リファレンス」の「DeleteAlarms」を参照してください。

`DescribeLogGroups`

次の例は、`DescribeLogGroups` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
```

```
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeLogGroups」を参照してください。

`DescribeSubscriptionFilters`

次の例は、`DescribeSubscriptionFilters` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeSubscriptionFilters」を参照してください。

GetQueryResults

次の例は、GetQueryResults を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetQueryResults」を参照してください。

PutSubscriptionFilter

次の例は、PutSubscriptionFilter を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutSubscriptionFilter」を参照してください。

StartLiveTail

次の例は、StartLiveTail を使用する方法を説明しています。

SDK for JavaScript (v3)

必要なファイルを含めます。

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-
cloudwatch-logs";
```

Live Tail セッションのイベントを処理します。

```
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)
    }
}
```

Live Tail セッションを開始します。

```
    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);
    }
```

一定時間が経過したら Live Tail セッションを停止します。

```
    /* Set a timeout to close the client. This will stop the Live Tail session. */
    setTimeout(function() {
        console.log("Client timeout");
        client.destroy();
      }, 10000);
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartLiveTail」を参照してください。

`StartQuery`

次の例は、`StartQuery` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartQuery」を参照してください。

シナリオ

大規模なクエリを実行する

次のコード例は、CloudWatch Logs を使用して 10,000 を超えるレコードをクエリする方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

これはエントリポイントです。

```
// 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}`,
);
```

これは、必要に応じてクエリを複数のステップに分割するクラスです。

```
// 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;
      },
    );
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - GetQueryResults
  - StartQuery

スケジュールされたイベントを使用した Lambda 関数の呼び出し

次のコード例は、Amazon EventBridge スケジュールされたイベントによって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- CloudWatch Logs

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

SDK for JavaScript (v3) を使用した CodeBuild の例

次のコード例は、CodeBuild で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック
- アクション

アクション

`CreateProject`

次の例は、`CreateProject` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

プロジェクトを作成します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateProject」を参照してください。

SDK for JavaScript (v3) を使用した Amazon Cognito ID の例

次のコード例は、Amazon Cognito Identity で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- シナリオ

シナリオ

Amazon Textract エクスプローラーアプリケーションを作成する

次のコード例は、インタラクティブアプリケーションを使用して Amazon Textract 出力を調べる方法を示しています。

SDK for JavaScript (v3)

を使用して、Amazon Textract を使用してドキュメントイメージからデータを抽出し、インタラクティブなウェブページに表示する React アプリケーション AWS SDK for JavaScript を構築する方法を示します。この例はウェブブラウザで実行され、認証情報に認証された Amazon Cognito ID が必要です。Amazon Simple Storage Service (Amazon S3) をストレージに使用し、通知のために、Amazon Simple Notification Service (Amazon SNS) トピックにサブスクライブした Amazon Simple Queue Service (Amazon SQS) キューをポーリングします。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

SDK for JavaScript (v3) を使用した Amazon Cognito ID プロバイダーの例

次のコード例は、Amazon Cognito ID プロバイダーで AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Amazon Cognito

次のコード例は、Amazon Cognito の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListUserPools」を参照してください。

トピック

- アクション

- シナリオ

アクション

`AdminGetUser`

次の例は、AdminGetUser を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

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

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AdminGetUser」を参照してください。

`AdminInitiateAuth`

次の例は、AdminInitiateAuth を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AdminInitiateAuth」を参照してください。

`AdminRespondToAuthChallenge`

次の例は、AdminRespondToAuthChallenge を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AdminRespondToAuthChallenge」を参照してください。

`AssociateSoftwareToken`

次の例は、`AssociateSoftwareToken` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AssociateSoftwareToken」を参照してください。

`ConfirmDevice`

次の例は、`ConfirmDevice` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ConfirmDevice」を参照してください。

`ConfirmSignUp`

次の例は、`ConfirmSignUp` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

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

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ConfirmSignUp」を参照してください。

`DeleteUser`

次の例は、`DeleteUser` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteUser」を参照してください。

`InitiateAuth`

次の例は、`InitiateAuth` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「InitiateAuth」を参照してください。

`ListUsers`

次の例は、`ListUsers` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

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

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListUsers」を参照してください。

ResendConfirmationCode

次の例は、ResendConfirmationCode を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

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

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ResendConfirmationCode」を参照してください。

RespondToAuthChallenge

次の例は、RespondToAuthChallenge を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「RespondToAuthChallenge」を参照してください。

SignUp

次の例は、SignUp を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SignUp」を参照してください。

`UpdateUserPool`

次の例は、`UpdateUserPool` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateUserPool」を参照してください。

`VerifySoftwareToken`

次の例は、`VerifySoftwareToken` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「VerifySoftwareToken」を参照してください。

シナリオ

Lambda 関数を使用して登録済みのユーザーを自動的に確認する

次のコード例は、Lambda 関数を使用して登録済みの Amazon Cognito ユーザーを確認する方法を示しています。

- PreSignUp トリガーの Lambda 関数を呼び出すようにユーザープールを設定します。
- Amazon Cognito でユーザーをサインアップする
- Lambda 関数は DynamoDB テーブルをスキャンし、登録済みのユーザーを自動的に確認します。
- 新しいユーザーとしてサインインし、リソースをクリーンアップします。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

インタラクティブな「シナリオ」実行を設定します。JavaScript (v3) の例では、シナリオランナーを共有して、複雑な例を効率化します。完全なソースコードは GitHub にあります。

```
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.",
  });
}
```

このシナリオでは、既知のユーザーを自動確認する方法を示します。サンプルのステップをオーケストレーションします。

```
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,
  );
```

これらは、他のシナリオと共有されるステップです。

```
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 },
);
```

Lambda 関数を使用する PreSignUp トリガーのハンドラー。

```
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);
};
```

CloudWatch Logs アクションのモジュール。

```
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];
  }
};
```

Amazon Cognito アクションのモジュール。

```
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];
  }
};
```

DynamoDB アクションのモジュール。

```
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];
  }
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

MFA を必要とするユーザープールによりユーザーをサインアップする

次のコードサンプルは、以下の操作方法を示しています。

- ユーザー名、パスワード、E メールアドレスでサインアップしてユーザーを確認します。
- MFA アプリケーションをユーザーに関連付けて、多要素認証を設定します。
- パスワードと MFA コードを使用してサインインします。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

最適なエクスペリエンスを得るには、GitHub リポジトリを複製してこの例を実行します。次のコードは、サンプルアプリケーション全体のサンプルを表しています。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - AdminGetUser
  - AdminInitiateAuth

- AdminRespondToAuthChallenge
- AssociateSoftwareToken
- ConfirmDevice
- ConfirmSignUp
- InitiateAuth
- ListUsers
- ResendConfirmationCode
- RespondToAuthChallenge
- SignUp
- VerifySoftwareToken

SDK for JavaScript (v3) を使用した Amazon Comprehend の例

次のコード例は、Amazon Comprehend で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- シナリオ

シナリオ

Amazon Transcribe ストリーミングアプリケーションを構築する

次のコード例は、ライブ音声をリアルタイムで記録、転写、翻訳し、結果を E メールで送信するアプリケーションを構築する方法を示しています。

SDK for JavaScript (v3)

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、転写、翻訳し、Amazon Simple Email Service (Amazon SES) を使用して結果を E メールで送信するアプリケーションを構築する方法について説明します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Amazon Lex chatbot を構築する

次のコード例は、ウェブサイトの訪問者を引き付けるチャットボットを作成する方法を示しています。

SDK for JavaScript (v3)

ウェブアプリケーション内に Amazon Lex chatbotを作成して、ウェブサイトの訪問者に対応することができます。

完全なソースコードとセットアップと実行の手順については、 デ AWS SDK for JavaScript ベロッパーガイドのAmazon Lexチャットボットの構築」の完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

顧客からのフィードバックを分析するアプリケーションの作成

次のコード例は、顧客のコメントカードを分析し、元の言語から翻訳し、顧客の感情を判断し、翻訳されたテキストから音声ファイルを生成するアプリケーションの作成方法を示しています。

SDK for JavaScript (v3)

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、GitHub のプロジェクトを参照してください。次の抜粋 AWS SDK for JavaScript は、Lambda 関数内で がどのように使用されるかを示しています。

```
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 };
};
```

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

SDK for JavaScript (v3) を使用した Amazon DocumentDB の例

次のコード例は、Amazon DocumentDB で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

サーバーレスサンプル

Amazon DocumentDB トリガーから Lambda 関数を呼び出す

次のコード例は、DocumentDB 変更ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DocumentDB ペイロードを取得し、レコードの内容を口グ記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で Amazon DocumentDB イベントの消費。

```
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));
};
```

TypeScript を使用して Lambda で Amazon DocumentDB イベントの消費。

```
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));
};
```

SDK for JavaScript (v3) を使用した DynamoDB の例

次のコード例は、DynamoDB で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello DynamoDB

次のコード例は、DynamoDB の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

での DynamoDB の使用の詳細については AWS SDK for JavaScript、JavaScript を使用した DynamoDB のプログラミング」を参照してください。

```
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;
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の「ListTables」を参照してください。

トピック

- 基本
- アクション
- シナリオ
- サーバーレスサンプル

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- 映画データを保持できるテーブルを作成する。
- テーブルに 1 つの映画を入れ、取得して更新する。
- サンプル JSON ファイルから映画データをテーブルに書き込む。
- 特定の年にリリースされた映画を照会する。
- 何年もの間にリリースされた映画をスキャンする。
- テーブルからムービーを削除し、テーブルを削除します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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 client.send(deleteCommand);
  log("Movie deleted.");

  /**
   * Upload a batch of movies.
   */

  log("Adding movies from local JSON file.");
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );
  const movies = JSON.parse(file.toString());
  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));
```

```
    const command = new BatchWriteCommand({
      RequestItems: {
        [tableName]: putRequests,
      },
    });

    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
```

```
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression. Scan
 will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

アクション

`BatchExecuteStatement`

次の例は、`BatchExecuteStatement` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

PartiQL を使用して項目のバッチを作成します。

```
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;
};
```

PartiQL を使用して項目のバッチを取得します。

```
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;
};
```

PartiQL を使用して項目のバッチを更新します。

```
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;
};
```

PartiQL を使用して項目のバッチを削除します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「BatchExecuteStatement」を参照してください。

`BatchGetItem`

次の例は、`BatchGetItem` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「BatchGet」を参照してください。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「BatchGetItem」を参照してください。

`BatchWriteItem`

次の例は、`BatchWriteItem` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「BatchWrite」を参照してください。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「BatchWriteItem」を参照してください。

`CreateTable`

次の例は、`CreateTable` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateTable」を参照してください。

DeleteItem

次の例は、DeleteItem を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「DeleteCommand」を参照してください。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteItem」を参照してください。

`DeleteTable`

次の例は、`DeleteTable` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の「DeleteTable」を参照してください。

`DescribeTable`

次の例は、`DescribeTable` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DescribeTable」を参照してください。

`DescribeTimeToLive`

次の例は、`DescribeTimeToLive` を使用する方法を説明しています。

SDK for JavaScript (v3)

AWS SDK for JavaScriptを使用して既存の DynamoDB テーブルの TTL 設定を記述します。

```
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');
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「DescribeTimeToLive」を参照してください。

`ExecuteStatement`

次の例は、`ExecuteStatement` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

PartiQL を使用して項目を作成します。

```
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;
};
```

PartiQL を使用して項目を取得します。

```
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;
};
```

PartiQL を使用して項目を更新します。

```
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;
};
```

PartiQL を使用して項目を削除します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ExecuteStatement」を参照してください。

GetItem

次の例は、GetItem を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「GetCommand」を参照してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetItem」を参照してください。

ListTables

次の例は、ListTables を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListTables」を参照してください。

`PutItem`

次の例は、`PutItem` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「PutCommand」を参照してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutItem」を参照してください。

`Query`

次の例は、`Query` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「QueryCommand」を参照してください。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「Query」を参照してください。

Scan

次の例は、Scan を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「ScanCommand」を参照してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Scan」を参照してください。

`UpdateItem`

次の例は、`UpdateItem` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「UpdateCommand」を参照してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

`UpdateTimeToLive`

次の例は、UpdateTimeToLive を使用する方法を説明しています。

SDK for JavaScript (v3)

既存の DynamoDB テーブルの TTL を有効にします。

```
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');
```

既存の DynamoDB テーブルの TTL を無効にします。

```
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');
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「UpdateTimeToLive」を参照してください。

シナリオ

DynamoDB テーブルにデータを送信するアプリケーションを構築する

次のコード例は、Amazon DynamoDB テーブルにデータを送信し、ユーザーがテーブルを更新したときに通知するアプリケーションを構築する方法を示しています。

SDK for JavaScript (v3)

この例では、ユーザーが Amazon DynamoDB テーブルにデータを送信し、Amazon Simple Notification Service (Amazon SNS) を使用して管理者にテキストメッセージを送信できるようにするアプリを構築する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- DynamoDB
- Amazon SNS

複数の値を 1 つの属性と比較する

次のコード例は、DynamoDB で複数の値を 1 つの属性と比較する方法を示しています。

- IN 演算子を使用して、複数の値を 1 つの属性と比較します。
- IN 演算子を複数の OR 条件と比較します。
- IN の使用によるパフォーマンスと式の複雑さのメリットを理解します。

SDK for JavaScript (v3)

を使用して、複数の値を単一の属性と比較します 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
    };
  }
}
```

複数の値と を比較する使用例 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);
  }
}
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - Query

- Scan

項目の TTL を条件付きで更新する

次のコード例は、項目の TTL を条件付きで更新する方法を示しています。

SDK for JavaScript (v3)

条件を指定して、テーブル内の既存の DynamoDB 項目の TTL を更新します。

```
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');
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

カウント式演算子

次のコード例は、DynamoDB で式演算子をカウントする方法を示しています。

- DynamoDB の演算子の制限 300 を理解します。
- 複雑な式での演算子をカウントする。
- 式を最適化して制限内に収まるようにします。

SDK for JavaScript (v3)

AWS SDK for JavaScriptを使用して式演算子のカウントをデモンストレーションします。

```
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;
}
```

でカウントする式演算子の使用例 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);
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

ウォームスループットを有効にしたテーブルを作成する

次のコード例は、ウォームスループットを有効にしてテーブルを作成する方法を示しています。

SDK for JavaScript (v3)

AWS SDK for JavaScriptを使用してウォームスループット設定を含む DynamoDB テーブルを作成します。

```
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
);
*/
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の「CreateTable」を参照してください。

TTL を含む項目を作成する

次のコード例は、TTL を使用して項目を作成する方法を示しています。

SDK for 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');
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutItem」を参照してください。

PartiQL DELETE を使用してデータを削除する

次のコード例は、PartiQL DELETE ステートメントを使用してデータを削除する方法を示しています。

SDK for JavaScript (v3)

PartiQL DELETE ステートメントを使用して、DynamoDB テーブルから項目を削除します 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");
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。

- BatchExecuteStatement
- ExecuteStatement

PartiQL INSERT を使用してデータを挿入する

次のコード例は、PartiQL INSERT ステートメントを使用してデータを挿入する方法を示しています。

SDK for JavaScript (v3)

PartiQL INSERT ステートメントを使用して、DynamoDB テーブルに項目を挿入します 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);
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - BatchExecuteStatement
  - ExecuteStatement

ブラウザからの Lambda 関数の呼び出し

次のコード例は、ブラウザから AWS Lambda 関数を呼び出す方法を示しています。

SDK for JavaScript (v3)

AWS Lambda 関数を使用して Amazon DynamoDB テーブルをユーザー選択で更新するブラウザベースのアプリケーションを作成できます。このアプリは v3 AWS SDK for JavaScript を使用します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Lambda

高度なクエリオペレーションを実行する

次のコード例は、DynamoDB で高度なクエリオペレーションを実行する方法を示しています。

- さまざまなフィルタリングと条件の手法を使用してテーブルをクエリします。
- 大きな結果セットのページ分割を実装します。
- 代替アクセスパターンにはグローバルセカンダリインデックスを使用します。

- アプリケーション要件に基づいて整合性コントロールを適用します。

SDK for JavaScript (v3)

を使用して、強力な整合性のある読み取りでクエリを実行します 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;
  }
}
```

でグローバルセカンダリインデックスを使用してクエリを実行します 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;
  }
}
```

を使用してページ分割でクエリを実行します 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 };
```

を使用して複雑なフィルターでクエリを実行します 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;
  }
}
```

を使用して動的に構築されたフィルター式でクエリを実行します 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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

リストオペレーションを実行する

次のコード例は、DynamoDB でリストオペレーションを実行する方法を示しています。

- リスト属性に要素を追加します。
- リスト属性から要素を削除します。
- インデックスでリスト内の特定の要素を更新します。
- リスト追加関数およびリストインデックス関数を使用します。

SDK for JavaScript (v3)

を使用してリストオペレーションをデモンストレーションします 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;
}
```

でのリストオペレーションの使用例 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;
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

マップオペレーションを実行する

次のコード例は、DynamoDB でマップオペレーションを実行する方法を示しています。

- マップ構造にネストされた属性を追加および更新します。
- マップから特定のフィールドを削除します。
- 深くネストされたマップ属性を操作します。

SDK for JavaScript (v3)

を使用してマップオペレーションをデモンストレーションします 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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

セットオペレーションを実行する

次のコード例は、DynamoDB でセットオペレーションを実行する方法を示しています。

- セット属性に要素を追加します。
- セット属性から要素を削除します。
- セットで ADD および DELETE オペレーションを使用します。

SDK for JavaScript (v3)

を使用してセットオペレーションをデモンストレーションします 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;
}
```

でのセットオペレーションの使用例 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);
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

PartiQL ステートメントのバッチを使用してテーブルにクエリを実行する

次のコードサンプルは、以下の操作方法を示しています。

- 複数の SELECT ステートメントを実行して、項目のバッチを取得する。
- 複数の INSERT ステートメントを実行して、項目のバッチを追加する。
- 複数の UPDATE ステートメントを実行して、項目のバッチを更新する。
- 複数の DELETE ステートメントを実行して、項目のバッチを削除する。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

PartiQL ステートメントのバッチを実行します。

```
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.");
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「BatchExecuteStatement」を参照してください。

PartiQL を使用してテーブルに対してクエリを実行する

次のコードサンプルは、以下の操作方法を示しています。

- SELECT ステートメントを実行して項目を取得する。

- INSERT 文を実行して項目を追加する。
- UPDATE ステートメントを使用して項目を更新する。
- DELETE ステートメントを実行して項目を削除する。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

単一の PartiQL ステートメントを実行します。

```
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.");
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ExecuteStatement」を参照してください。

グローバルセカンダリインデックスを使用してテーブルをクエリする

次のコード例は、グローバルセカンダリインデックスを使用してテーブルをクエリする方法を示しています。

- プライマリキーを使用して DynamoDB テーブルをクエリします。
- 代替アクセスパターンのグローバルセカンダリインデックス (GSI) をクエリします。
- テーブルクエリと GSI クエリを比較します。

SDK for JavaScript (v3)

プライマリキーを使用して DynamoDB テーブルをクエリします 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;
  }
}
```

を使用して DynamoDB グローバルセカンダリインデックス (GSI) をクエリします 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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

begins_with 条件を使用してテーブルをクエリする

次のコード例は、 begins_with 条件を使用してテーブルをクエリする方法を示しています。

- キー条件式で begins_with 関数を使用します。
- ソートキーのプレフィックスパターンに基づいて項目をフィルタリングします。

SDK for JavaScript (v3)

AWS SDK for JavaScriptでソートキーの begins_with 条件を使用して DynamoDB テーブルをクエリします。

```
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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

日付範囲を使用してテーブルをクエリする

次のコード例は、ソートキーの日付範囲を使用してテーブルをクエリする方法を示しています。

- 特定の日付範囲内の項目をクエリします。
- 日付形式のソートキーで比較演算子を使用します。

SDK for JavaScript (v3)

を使用して、日付範囲内の項目について DynamoDB テーブルをクエリします 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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

複雑なフィルター式を使用してテーブルをクエリする

次のコード例は、複雑なフィルター式を使用してテーブルをクエリする方法を示しています。

- 複雑なフィルター式をクエリ結果に適用します。
- 論理演算子を使用して複数の条件を組み合わせます。
- キー以外の属性に基づいて項目をフィルタリングします。

SDK for JavaScript (v3)

を使用して、複雑なフィルター式で DynamoDB テーブルをクエリします 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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

動的フィルター式を使用してテーブルをクエリする

次のコード例は、動的フィルター式を使用してテーブルをクエリする方法を示しています。

- ランタイムにフィルター式を動的に構築します。
- ユーザー入力またはアプリケーション状態に基づいてフィルター条件を構築します。
- 条件付きでフィルター条件を追加または削除します。

SDK for JavaScript (v3)

を使用して動的に構築されたフィルター式で DynamoDB テーブルをクエリします 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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

ネストされた属性を使用してテーブルをクエリする

次のコード例は、ネストされた属性を持つテーブルをクエリする方法を示しています。

- DynamoDB 項目のネストされた属性を使用してアクセスおよびフィルタリングします。
- ネストされた要素を参照するには、ドキュメントパス式を使用します。

SDK for JavaScript (v3)

を使用して、ネストされた属性を持つ DynamoDB テーブルをクエリします 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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

ページ分割を使用してテーブルをクエリする

次のコード例は、ページ分割を使用してテーブルをクエリする方法を示しています。

- DynamoDB クエリ結果のページ分割を実装します。
- LastEvaluatedKey を使用して後続のページを取得します。
- Limit パラメータを使用して、ページあたりの項目数を制御します。

SDK for JavaScript (v3)

を使用してページ分割で DynamoDB テーブルをクエリします 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 };
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

強力な整合性のある読み込みを使用してテーブルをクエリする

次のコード例は、強力な整合性のある読み込みでテーブルをクエリする方法を示しています。

- DynamoDB クエリの整合性レベルを設定します。
- 強力な整合性のある読み込みを使用して最新のデータを取得します。
- 結果整合性と強力な整合性のトレードオフを理解します。

SDK for JavaScript (v3)

を使用して、設定可能な読み取り整合性で DynamoDB テーブルをクエリします 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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

PartiQL SELECT を使用してデータをクエリする

次のコード例は、PartiQL SELECT ステートメントを使用してデータをクエリする方法を示しています。

SDK for JavaScript (v3)

PartiQL SELECT ステートメントを使用して DynamoDB テーブルから項目をクエリします 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");
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - BatchExecuteStatement
  - ExecuteStatement

TTL 項目のクエリ

次のコード例は、TTL 項目をクエリする方法を示しています。

SDK for JavaScript (v3)

を使用して DynamoDB テーブルで TTL 項目を収集するためにフィルタリングされた式をクエリします 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');
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

日付と時刻のパターンを使用してテーブルをクエリする

次のコード例は、日付と時刻のパターンを使用してテーブルをクエリする方法を示しています。

- DynamoDB に日付/時刻値を保存してクエリします。
- ソートキーを使用して日付範囲クエリを実装します。
- 有効なクエリを行うために日付文字列をフォーマットします。

SDK for JavaScript (v3)

でソートキーの日付範囲を使用してクエリを実行します 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;
  }
}
```

で日時変数を使用してクエリを実行します 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;
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

式順序の更新を理解する

次のコード例は、更新式の順序を理解する方法を示しています。

- DynamoDB が更新式を処理する方法について説明します。
- 更新式のオペレーションの順序を理解します。
- 式の評価を理解することで、予期しない結果を回避できます。

SDK for JavaScript (v3)

を使用して、式の更新順序をデモンストレーションします 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
  };
}
```

での更新式の順序の使用例 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);
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

テーブルのウォームスループット設定を更新する

次のコード例は、テーブルのウォームスループット設定を更新する方法を示しています。

SDK for JavaScript (v3)

AWS SDK for JavaScriptを使用して既存の DynamoDB テーブルのウォームスループット設定を更新します。

```
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
);
*/
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateTable」を参照してください。

項目を TTL を更新する

次のコード例は、項目の TTL を更新する方法を示しています。

SDK for 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');
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

PartiQL UPDATE を使用してデータを更新する

次のコード例は、PartiQL UPDATE ステートメントを使用してデータを更新する方法を示しています。

SDK for JavaScript (v3)

PartiQL UPDATE ステートメントを使用して DynamoDB テーブルの項目を更新します 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(),
    },
  ]);
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - BatchExecuteStatement
  - ExecuteStatement

API Gateway を使用して Lambda 関数を呼び出す

次のコード例は、Amazon API Gateway によって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

Lambda JavaScript ランタイム API を使用して AWS Lambda 関数を作成する方法を示します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、Amazon API Gateway によって呼び出される Lambda 関数を作成する方法を示します。この関数は、Amazon DynamoDB テーブルをスキャンして、Amazon Simple Notification Service (Amazon SNS) を使用して、従業員に年間の記念日を祝福するテキストメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

アトミックカウンターオペレーションを使用する

次のコード例は、DynamoDB でアトミックカウンターオペレーションを使用する方法を示しています。

- ADD および SET オペレーションを使用してカウンターをアトミックに増分します。
- 存在しない可能性のあるカウンターを安全に増分します。
- カウンターオペレーションに楽観的ロックを実装します。

SDK for JavaScript (v3)

を使用してアトミックカウンターオペレーションをデモンストレーションします 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;
}
```

でのアトミックカウンターオペレーションの使用例 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);
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

条件付きオペレーションを使用する

次のコード例は、DynamoDB で条件付きオペレーションを使用する方法を示しています。

- データの上書きを防ぐため、条件付き書き込みを実装します。
- 条件式を使用してビジネスルールを適用します。
- 条件付きチェックの失敗を適切に処理します。

SDK for JavaScript (v3)

を使用して条件付きオペレーションをデモンストレーションします 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;
}
```

での条件付きオペレーションの使用例 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);
  }
}
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - DeleteItem
  - PutItem
  - UpdateItem

式の属性名を使用する

次のコード例は、DynamoDB で式属性名を使用する方法を示しています。

- DynamoDB 式で予約語を操作します。
- 式の属性名のプレースホルダーを使用します。
- 属性名の特殊文字を処理します。

SDK for JavaScript (v3)

を使用して式の属性名をデモンストレーションします 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;
}
```

を使用した式の属性名の使用例 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);
  }
}
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - Query
  - UpdateItem

スケジュールされたイベントを使用した Lambda 関数の呼び出し

次のコード例は、Amazon EventBridge スケジュールされたイベントによって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

サーバーレスサンプル

DynamoDB トリガーから Lambda 関数を呼び出す

次のコード例は、DynamoDB ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DynamoDB ペイロードを取得し、レコードの内容をログ記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で DynamoDB イベントの消費。

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

TypeScript を使用した Lambda での DynamoDB イベントの消費。

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

DynamoDB トリガーで Lambda 関数のバッチアイテムの失敗をレポートする

次のコード例は、DynamoDB ストリームからイベントを受信する Lambda 関数に部分的なバッチレスポンスを実装する方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
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: [] };
};
```

TypeScript を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
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 };
};
```

SDK for JavaScript (v3) を使用した Amazon EC2 の例

次のコード例は、Amazon EC2 で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Amazon EC2

以下のコード例は、Amazon EC2 の利用開始方法を表示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeSecurityGroups」を参照してください。

トピック

- 基本
- アクション
- シナリオ

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- キーペアとセキュリティグループを作成します。
- Amazon マシンイメージ (AMI)と互換性のあるインスタンスタイプを選択し、インスタンスを作成します。
- インスタンスを停止し、再起動します。
- Elastic IP アドレスをインスタンスに関連付ける。
- SSH を使用してインスタンスに接続し、リソースをクリーンアップします。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

このファイルには、EC2 で使用される一般的なアクションのリストが含まれています。ステップは、インタラクティブな例の実行を簡素化するシナリオフレームワークを使用して構築されます。完全なコンテキストについては、GitHub リポジトリにアクセスしてください。

```
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,
  },
);
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - AllocateAddress
  - AssociateAddress
  - AuthorizeSecurityGroupIngress
  - CreateKeyPair
  - CreateSecurityGroup
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DescribeImages
  - DescribeInstanceTypes
  - DescribeInstances
  - DescribeKeyPairs
  - DescribeSecurityGroups
  - DisassociateAddress
  - ReleaseAddress

- RunInstances
- StartInstances
- StopInstances
- TerminateInstances
- UnmonitorInstances

アクション

`AllocateAddress`

次の例は、`AllocateAddress` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AllocateAddress」を参照してください。

`AssociateAddress`

次の例は、`AssociateAddress` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AssociateAddress」を参照してください。

AuthorizeSecurityGroupIngress

次の例は、AuthorizeSecurityGroupIngress を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AuthorizeSecurityGroupIngress」を参照してください。

`CreateKeyPair`

次の例は、`CreateKeyPair` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateKeyPair」を参照してください。

`CreateLaunchTemplate`

次の例は、`CreateLaunchTemplate` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
  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,
        },
      }),
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateLaunchTemplate」を参照してください。

`CreateSecurityGroup`

次の例は、`CreateSecurityGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateSecurityGroup」を参照してください。

`DeleteKeyPair`

次の例は、`DeleteKeyPair` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteKeyPair」を参照してください。

`DeleteLaunchTemplate`

次の例は、`DeleteLaunchTemplate` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
  await client.send(
    new DeleteLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
    }),
  );
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteLaunchTemplate」を参照してください。

`DeleteSecurityGroup`

次の例は、`DeleteSecurityGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteSecurityGroup」を参照してください。

`DescribeAddresses`

次の例は、`DescribeAddresses` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeAddresses」を参照してください。

`DescribeIamInstanceProfileAssociations`

次の例は、`DescribeIamInstanceProfileAssociations` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeIamInstanceProfileAssociations」を参照してください。

`DescribeImages`

次の例は、`DescribeImages` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeImages」を参照してください。

`DescribeInstanceTypes`

次の例は、`DescribeInstanceTypes` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeInstanceTypes」を参照してください。

`DescribeInstances`

次の例は、`DescribeInstances` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeInstances」を参照してください。

`DescribeKeyPairs`

次の例は、`DescribeKeyPairs` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeKeyPairs」を参照してください。

DescribeRegions

次の例は、DescribeRegions を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeRegions」を参照してください。

`DescribeSecurityGroups`

次の例は、`DescribeSecurityGroups` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeSecurityGroups」を参照してください。

`DescribeSubnets`

次の例は、`DescribeSubnets` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    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"] },
      ],
    }),
  );
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeSubnets」を参照してください。

`DescribeVpcs`

次の例は、`DescribeVpcs` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeVpcs」を参照してください。

`DisassociateAddress`

次の例は、`DisassociateAddress` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DisassociateAddress」を参照してください。

`MonitorInstances`

次の例は、`MonitorInstances` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「MonitorInstances」を参照してください。

`RebootInstances`

次の例は、`RebootInstances` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「RebootInstances」を参照してください。

`ReleaseAddress`

次の例は、`ReleaseAddress` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ReleaseAddress」を参照してください。

`ReplaceIamInstanceProfileAssociation`

次の例は、`ReplaceIamInstanceProfileAssociation` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ReplaceIamInstanceProfileAssociation」を参照してください。

`RunInstances`

次の例は、`RunInstances` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「RunInstances」を参照してください。

`StartInstances`

次の例は、`StartInstances` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
```

```
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartInstances」を参照してください。

`StopInstances`

次の例は、`StopInstances` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StopInstances」を参照してください。

`TerminateInstances`

次の例は、`TerminateInstances` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「TerminateInstances」を参照してください。

`UnmonitorInstances`

次の例は、UnmonitorInstances を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UnmonitorInstances」を参照してください。

シナリオ

レジリエントなサービスの構築と管理

次のコード例は、本、映画、曲のレコメンデーションを返す負荷分散型ウェブサービスの作成方法を示しています。この例は、障害に対するサービスの対応方法と、障害発生時の耐障害性を高めるためにサービスを再構築する方法を示しています。

- Amazon EC2 Auto Scaling グループを使用して、起動テンプレートに基づいて Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを作成し、インスタンス数を所定の範囲内に維持します。
- Elastic Load Balancing で HTTP リクエストを処理して配信します。
- Auto Scaling グループ内のインスタンスの状態を監視し、正常なインスタンスにのみリクエストを転送します。
- 各 EC2 インスタンスで Python ウェブサーバーを実行して HTTP リクエストを処理します。ウェブサーバーはレコメンデーションとヘルスチェックを返します。
- Amazon DynamoDB テーブルを使用してレコメンデーションサービスをシミュレートできます。
- AWS Systems Manager パラメータを更新して、リクエストとヘルスチェックに対するウェブサーバーの応答を制御します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
#!/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.",
  });
}
```

すべてのリソースをデプロイするための手順を作成します。

```
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,
];
```

デモを実行するための手順を作成します。

```
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;
}
```

すべてのリソースを破棄するための手順を作成します。

```
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.`);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - 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 - SDK for JavaScript (v3) を使用したバージョン 2 の例

次のコード例は、Elastic Load Balancing - バージョン 2 で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Elastic Load Balancing

次のコード例は、Elastic Load Balancing の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeLoadBalancers」を参照してください。

トピック

- アクション
- シナリオ

アクション

`CreateListener`

次の例は、`CreateListener` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateListener」を参照してください。

`CreateLoadBalancer`

次の例は、`CreateLoadBalancer` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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] },
);
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateLoadBalancer」を参照してください。

`CreateTargetGroup`

次の例は、`CreateTargetGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateTargetGroup」を参照してください。

`DeleteLoadBalancer`

次の例は、`DeleteLoadBalancer` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.");
  }
});
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteLoadBalancer」を参照してください。

`DeleteTargetGroup`

次の例は、`DeleteTargetGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteTargetGroup」を参照してください。

`DescribeLoadBalancers`

次の例は、`DescribeLoadBalancers` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeLoadBalancers」を参照してください。

`DescribeTargetGroups`

次の例は、`DescribeTargetGroups` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeTargetGroups」を参照してください。

`DescribeTargetHealth`

次の例は、`DescribeTargetHealth` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeTargetHealth」を参照してください。

シナリオ

レジリエントなサービスの構築と管理

次のコード例は、本、映画、曲のレコメンデーションを返す負荷分散型ウェブサービスの作成方法を示しています。この例は、障害に対するサービスの対応方法と、障害発生時の耐障害性を高めるためにサービスを再構築する方法を示しています。

- Amazon EC2 Auto Scaling グループを使用して、起動テンプレートに基づいて Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを作成し、インスタンス数を所定の範囲内に維持します。
- Elastic Load Balancing で HTTP リクエストを処理して配信します。

- Auto Scaling グループ内のインスタンスの状態を監視し、正常なインスタンスにのみリクエストを転送します。
- 各 EC2 インスタンスで Python ウェブサーバーを実行して HTTP リクエストを処理します。ウェブサーバーはレコメンデーションとヘルスチェックを返します。
- Amazon DynamoDB テーブルを使用してレコメンデーションサービスをシミュレートできます。
- AWS Systems Manager パラメータを更新して、リクエストとヘルスチェックに対するウェブサーバーの応答を制御します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
#!/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.",
  });
}
```

すべてのリソースをデプロイするための手順を作成します。

```
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,
];
```

デモを実行するための手順を作成します。

```
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;
}
```

すべてのリソースを破棄するための手順を作成します。

```
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.`);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - 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 SDK for JavaScript (v3) を使用した の例

次のコード例は、 で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています AWS Entity Resolution。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

こんにち AWS Entity Resolutionは

次のコード例は、 AWS Entity Resolutionの使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のlistMatchingWorkflows」を参照してください。

トピック

- アクション

アクション

`CheckWorkflowStatus`

次の例は、`CheckWorkflowStatus` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
//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 ({ workflowName, jobId }) => {
  const getMatchingJobParams = {
    workflowName: `${data.inputs.workflowName}`,
    jobId: `${data.inputs.jobId}`,
  };
  try {
    const command = new GetMatchingJobCommand(getMatchingJobParams);
    const response = await erClient.send(command);
    console.log(`Job status: ${response.status}`);
  } catch (error) {
    console.log("error ", error.message);
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCheckWorkflowStatus」を参照してください。

`CreateMatchingWorkflow`

次の例は、`CreateMatchingWorkflow` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCreateMatchingWorkflow」を参照してください。

`CreateSchemaMapping`

次の例は、`CreateSchemaMapping` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCreateSchemaMapping」を参照してください。

`DeleteMatchingWorkflow`

次の例は、`DeleteMatchingWorkflow` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeleteMatchingWorkflow」を参照してください。

`DeleteSchemaMapping`

次の例は、`DeleteSchemaMapping` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeleteSchemaMapping」を参照してください。

`GetMatchingJob`

次の例は、`GetMatchingJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のGetMatchingJob」を参照してください。

GetSchemaMapping

次の例は、GetSchemaMapping を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のGetSchemaMapping」を参照してください。

`ListSchemaMappings`

次の例は、`ListSchemaMappings` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
//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;
    }
  }
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のListSchemaMappings」を参照してください。

`StartMatchingJob`

次の例は、`StartMatchingJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のStartMatchingJob」を参照してください。

`TagEntityResource`

次の例は、`TagEntityResource` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」の「TagEntityResource」を参照してください。

SDK for JavaScript (v3) を使用した EventBridge の例

次のコード例は、EventBridge で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション
- シナリオ

アクション

PutEvents

次の例は、PutEvents を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutEvents」を参照してください。

`PutRule`

次の例は、`PutRule` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutRule」を参照してください。

PutTargets

次の例は、PutTargets を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutTargets」を参照してください。

シナリオ

スケジュールされたイベントを使用した Lambda 関数の呼び出し

次のコード例は、Amazon EventBridge スケジュールされたイベントによって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda

- Amazon SNS

AWS Glue SDK for JavaScript (v3) を使用した例

次のコード例は、 で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています AWS Glue。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

こんにち AWS Glueは

次のコード例は、 AWS Glueの使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListJobs」を参照してください。

トピック

- 基本
- アクション

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- パブリック Amazon S3 バケットをクロールし、CSV 形式のメタデータのデータベースを生成するクローラーを作成する。
- のデータベースとテーブルに関する情報を一覧表示します AWS Glue Data Catalog。
- S3 バケットから CSV 形式のデータを抽出するジョブを作成し、そのデータを変換して JSON 形式の出力を別の S3 バケットにロードする。
- ジョブ実行に関する情報を一覧表示し、変換されたデータを表示してリソースをクリーンアップする。

詳細については、「チュートリアル: AWS Glue Studio の開始方法」を参照してください。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

パブリックの Amazon Simple Storage Service (Amazon S3) バケットをクロールし、検出した CSV 形式データを記述するメタデータデータベースを生成するクローラーを作成して実行します。

```
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 };
  };
```

のデータベースとテーブルに関する情報を一覧表示します 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 };
  };
```

ソース Amazon S3 バケットから CSV 形式データを抽出し、フィールドを削除して名前を変更することで変換し、JSON 形式の出力を別の Amazon S3 バケットにロードするジョブを作成して実行します。

```
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 };
  };
```

ジョブ実行に関する情報を一覧表示し、変換されたデータの一部を表示します。

```
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 };
  };
```

デモによって作成されたすべてのリソースを削除します。

```
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 };
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase

- GetDatabases
- GetJob
- GetJob
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

アクション

`CreateCrawler`

次の例は、`CreateCrawler` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateCrawler」を参照してください。

`CreateJob`

次の例は、`CreateJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateJob」を参照してください。

DeleteCrawler

次の例は、DeleteCrawler を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteCrawler」を参照してください。

DeleteDatabase

次の例は、DeleteDatabase を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const deleteDatabase = (databaseName) => {
```

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

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteDatabase」を参照してください。

`DeleteJob`

次の例は、DeleteJob を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteJob」を参照してください。

`DeleteTable`

次の例は、`DeleteTable` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

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

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の「DeleteTable」を参照してください。

`GetCrawler`

次の例は、`GetCrawler` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getCrawler = (name) => {
```

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

  const command = new GetCrawlerCommand({
    Name: name,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetCrawler」を参照してください。

`GetDatabase`

次の例は、`GetDatabase` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetDatabase」を参照してください。

GetDatabases

次の例は、GetDatabases を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetDatabases」を参照してください。

GetJob

次の例は、GetJob を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getJob = (jobName) => {
  const client = new GlueClient({});
```

```
  const command = new GetJobCommand({
    JobName: jobName,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetJob」を参照してください。

GetJobRun

次の例は、GetJobRun を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetJobRun」を参照してください。

GetJobRuns

次の例は、GetJobRuns を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetJobRuns」を参照してください。

`GetTables`

次の例は、`GetTables` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });
```

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetTables」を参照してください。

`ListJobs`

次の例は、`ListJobs` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListJobs」を参照してください。

`StartCrawler`

次の例は、`StartCrawler` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartCrawler」を参照してください。

`StartJobRun`

次の例は、StartJobRun を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartJobRun」を参照してください。

SDK for JavaScript (v3) を使用した HealthImaging の例

次のコード例は、HealthImaging で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello HealthImaging

次のコード例では、HealthImaging の使用を開始する方法を示しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListDatastores」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

トピック

- アクション
- シナリオ

アクション

`CopyImageSet`

次の例は、`CopyImageSet` を使用する方法を説明しています。

SDK for JavaScript (v3)

イメージセットをコピーするためのユーティリティ関数。

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
```

```
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 image set.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = "",
  force = false,
  copySubsets = [],
) => {
  try {
    const params = {
      datastoreId: datastoreId,
      sourceImageSetId: imageSetId,
      copyImageSetInformation: {
        sourceImageSet: { latestVersionId: sourceVersionId },
      },
      force: force,
    };
    if (destinationImageSetId !== "" && destinationVersionId !== "") {
      params.copyImageSetInformation.destinationImageSet = {
        imageSetId: destinationImageSetId,
        latestVersionId: destinationVersionId,
      };
    }

    if (copySubsets.length > 0) {
      let copySubsetsJson;
      copySubsetsJson = {
        SchemaVersion: 1.1,
        Study: {
          Series: {
            imageSetId: {
              Instances: {},
            },
          },
```

```
        },
      };

      for (let i = 0; i < copySubsets.length; i++) {
        copySubsetsJson.Study.Series.imageSetId.Instances[copySubsets[i]] = {};
      }

      params.copyImageSetInformation.dicomCopies = copySubsetsJson;
    }

    const response = await medicalImagingClient.send(
      new CopyImageSetCommand(params),
    );
    console.log(response);
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //      },
    //       datastoreId: 'xxxxxxxxxxxxx',
    //       destinationImageSetProperties: {
    //             createdAt: 2023-09-27T19:46:21.824Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       },
    //       sourceImageSetProperties: {
    //             createdAt: 2023-09-22T14:49:26.427Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
    //             latestVersionId: '4',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //     }
```

```
    // }
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

コピー先を指定せずにイメージセットをコピーします。

```
  await copyImageSet(
    "1234567890123456789012345678901 2",
    "1234567890123456789012345678901 2",
    "1",
  );
```

コピー先を指定してイメージセットをコピーします。

```
  await copyImageSet(
    "1234567890123456789012345678901 2",
    "1234567890123456789012345678901 2",
    "1",
    "1234567890123456789012345678901 2",
    "1",
    false,
  );
```

イメージセットのサブセットを送信先にコピーし、コピーを強制します。

```
  await copyImageSet(
    "1234567890123456789012345678901 2",
    "1234567890123456789012345678901 2",
    "1",
    "1234567890123456789012345678901 2",
    "1",
    true,
```

```
    ["12345678901234567890123456789012", "11223344556677889900112233445566"],
  );
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CopyImageSet」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`CreateDatastore`

次の例は、`CreateDatastore` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateDatastore」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`DeleteDatastore`

次の例は、`DeleteDatastore` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteDatastore」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`DeleteImageSet`

次の例は、`DeleteImageSet` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteImageSet」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`GetDICOMImportJob`

次の例は、`GetDICOMImportJob` を使用する方法を説明しています。

SDK for 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: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetDICOMImportJob」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

GetDatastore

次の例は、GetDatastore を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetDatastore」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`GetImageFrame`

次の例は、`GetImageFrame` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetImageFrame」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`GetImageSet`

次の例は、`GetImageSet` を使用する方法を説明しています。

SDK for 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/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetImageSet」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

GetImageSetMetadata

次の例は、GetImageSetMetadata を使用する方法を説明しています。

SDK for JavaScript (v3)

イメージセットのメタデータを取得するためのユーティリティ関数。

```
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 = "xxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxx",
  versionID = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params),
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
```

```
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

イメージセットのメタデータをバージョンなしで取得します。

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
    );
  } catch (err) {
    console.log("Error", err);
  }
```

イメージセットのメタデータをバージョン付きで取得します。

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1",
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetImageSetMetadata」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`ListDICOMImportJobs`

次の例は、`ListDICOMImportJobs` を使用する方法を説明しています。

SDK for 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: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListDICOMImportJobs」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`ListDatastores`

次の例は、`ListDatastores` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListDatastores」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`ListImageSetVersions`

次の例は、`ListImageSetVersions` を使用する方法を説明しています。

SDK for 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: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListImageSetVersions」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`ListTagsForResource`

次の例は、`ListTagsForResource` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListTagsForResource」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`SearchImageSets`

次の例は、`SearchImageSets` を使用する方法を説明しています。

SDK for JavaScript (v3)

イメージセットを検索するためのユーティリティ関数。

```
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;
};
```

ユースケース #1: EQUAL 演算子。

```
  const datastoreId = "12345678901234567890123456789012";

  try {
    const searchCriteria = {
      filters: [
        {
          values: [{ DICOMPatientId: "1234567" }],
          operator: "EQUAL",
        },
      ],
    };

    await searchImageSets(datastoreId, searchCriteria);
  } catch (err) {
    console.error(err);
  }
```

ユースケース #2: DICOMStudyDate と DICOMStudyTime を使用する BETWEEN 演算子。

```
  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);
  }
```

ユースケース #3: createdAt を使用する BETWEEN 演算子。タイムスタディは以前に永続化されています。

```
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);
}
```

ユースケース #4: DICOMSeriesInstanceUID の EQUAL 演算子と updatedAt の BETWEEN 演算子、および updatedAt フィールドの ASC 順のソートレスポンス。

```
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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SearchImageSets」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`StartDICOMImportJob`

次の例は、`StartDICOMImportJob` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartDICOMImportJob」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`TagResource`

次の例は、`TagResource` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「TagResource」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`UntagResource`

次の例は、`UntagResource` を使用する方法を説明しています。

SDK for 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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UntagResource」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`UpdateImageSetMetadata`

次の例は、`UpdateImageSetMetadata` を使用する方法を説明しています。

SDK for 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);
  }
};
```

ユースケース #1: 属性を挿入または更新し、強制的に更新します。

```
    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,
    );
```

ユースケース #2: 属性を削除します。

```
    // 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,
);
```

ユースケース #3: インスタンスを削除します。

```
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,
  );
```

ユースケース #4: 以前のバージョンに戻します。

```
  const updateMetadata = {
    revertToVersionId: "1",
  };

  await updateImageSetMetadata(
    datastoreID,
    imageSetID,
    versionID,
    updateMetadata,
  );
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateImageSetMetadata」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

シナリオ

画像セットと画像フレームを使い始めます

次のコード例は、HealthImaging で DICOM ファイルをインポートし、イメージフレームをダウンロードする方法を示しています。

実装はコマンドラインアプリケーションとして構造化されています。

- DICOM インポートするためのリソースのセットアップ
- DICOM ファイルをデータストアへのインポート。
- インポートジョブの画像セット ID の取得。

- インポートジョブの画像フレーム ID の取得。
- イメージフレームをダウンロード、デコード、および検証します。
- リソースをクリーンアップします。

SDK for JavaScript (v3)

ステップをオーケストレーションします (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]",
  });
}
```

リソース () をデプロイします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 },
);
```

DICOM ファイル () をコピーします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.`,
);
```

データストア () へのインポートを開始します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}`,
);
```

イメージセット IDs (image-set-steps.js -) を取得します。

```
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")}`,
);
```

イメージフレーム IDs () を取得します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;
  },
);
```

イメージフレーム () を確認しますverify-steps.js。AWS HealthImaging Pixel データ検証ライブラリが検証に使用されました。

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

リソース () を破棄します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,
  },
);
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - DeleteImageSet
  - GetDICOMImportJob
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - StartDICOMImportJob

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

データストアにタグを付ける

次のコード例は、HealthImaging データストアにタグを付ける方法を示しています。

SDK for JavaScript (v3)

データストアにタグを付けます。

```
  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);
  }
```

リソースにタグを付けるためのユーティリティ関数。

```
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;
};
```

データストアのタグを一覧表示します。

```
  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);
  }
```

リソースのタグを一覧表示するユーティリティ関数。

```
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;
};
```

データストアのタグを解除するには

```
  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);
  }
```

リソースのタグを解除するユーティリティ関数。

```
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;
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

イメージセットにタグを付ける

次のコード例は、HealthImaging イメージセットにタグを付ける方法を示しています。

SDK for JavaScript (v3)

イメージセットにタグを付けるには

```
  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);
  }
```

リソースにタグを付けるためのユーティリティ関数。

```
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;
};
```

イメージセットのタグを一覧表示します。

```
  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);
  }
```

リソースのタグを一覧表示するユーティリティ関数。

```
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;
};
```

イメージセットのタグを解除します。

```
  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);
  }
```

リソースのタグを解除するユーティリティ関数。

```
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;
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

SDK for JavaScript (v3) を使用した IAM の例

次のコード例は、IAM で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

IAM へようこそ

次のコード例は、IAM の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListPolicies」を参照してください。

トピック

- 基本
- アクション
- シナリオ

基本

基本を学ぶ

次のコードサンプルは、ユーザーを作成してロールを割り当てる方法を示しています。

> **Warning**
>
> セキュリティリスクを避けるため、専用ソフトウェアの開発や実際のデータを扱うときは、IAM ユーザーを認証に使用しないでください。代わりに、AWS IAM Identity Center などの ID プロバイダーとのフェデレーションを使用してください。

- 権限のないユーザーを作成します。
- 指定したアカウントに Amazon S3 バケットへのアクセス権限を付与するロールを作成します。
- ユーザーにロールを引き受けさせるポリシーを追加します。
- ロールを引き受け、一時的な認証情報を使用して S3 バケットを一覧表示しリソースをクリーンアップします。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

IAM ユーザーと、Amazon S3 バケットを一覧表示するアクセス権限を付与するロールを作成します。ユーザーには、ロールの引き受けのみ権限があります。ロールを引き受けた後、一時的な認証情報を使用してアカウントのバケットを一覧表示します。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

アクション

`AttachRolePolicy`

次の例は、`AttachRolePolicy` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ポリシーをアタッチします。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AttachRolePolicy」を参照してください。

`CreateAccessKey`

次の例は、`CreateAccessKey` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アクセスキーを作成します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateAccessKey」を参照してください。

`CreateAccountAlias`

次の例は、`CreateAccountAlias` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アカウントエイリアスを作成します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateAccountAlias」を参照してください。

`CreateGroup`

次の例は、`CreateGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateGroup」を参照してください。

`CreateInstanceProfile`

次の例は、`CreateInstanceProfile` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateInstanceProfile」を参照してください。

`CreatePolicy`

次の例は、`CreatePolicy` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ポリシーを作成します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreatePolicy」を参照してください。

`CreateRole`

次の例は、`CreateRole` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ロールを作成します。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateRole」を参照してください。

`CreateSAMLProvider`

次の例は、`CreateSAMLProvider` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateSAMLProvider」を参照してください。

`CreateServiceLinkedRole`

次の例は、`CreateServiceLinkedRole` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

サービスにリンクされたロールを作成します。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateServiceLinkedRole」を参照してください。

CreateUser

次の例は、CreateUser を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ユーザーを作成します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateUser」を参照してください。

`DeleteAccessKey`

次の例は、`DeleteAccessKey` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アクセスキーを削除します

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteAccessKey」を参照してください。

`DeleteAccountAlias`

次の例は、`DeleteAccountAlias` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アカウントエイリアスを削除します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteAccountAlias」を参照してください。

`DeleteGroup`

次の例は、`DeleteGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteGroup」を参照してください。

DeleteInstanceProfile

次の例は、DeleteInstanceProfile を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteInstanceProfile」を参照してください。

DeletePolicy

次の例は、DeletePolicy を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ポリシーを削除します。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeletePolicy」を参照してください。

`DeleteRole`

次の例は、`DeleteRole` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ロールを削除します。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteRole」を参照してください。

`DeleteRolePolicy`

次の例は、DeleteRolePolicy を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteRolePolicy」を参照してください。

`DeleteSAMLProvider`

次の例は、`DeleteSAMLProvider` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteSAMLProvider」を参照してください。

`DeleteServerCertificate`

次の例は、`DeleteServerCertificate` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

サーバー証明書を削除します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteServerCertificate」を参照してください。

DeleteServiceLinkedRole

次の例は、DeleteServiceLinkedRole を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteServiceLinkedRole」を参照してください。

DeleteUser

次の例は、DeleteUser を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ユーザーを削除。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteUser」を参照してください。

`DetachRolePolicy`

次の例は、`DetachRolePolicy` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ポリシーをデタッチします。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DetachRolePolicy」を参照してください。

`GetAccessKeyLastUsed`

次の例は、`GetAccessKeyLastUsed` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アクセスキーを取得します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript SDK for Rust API リファレンス」の「GetAccessKeyLastUsed」を参照してください。

`GetAccountPasswordPolicy`

次の例は、`GetAccountPasswordPolicy` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アカウントのパスワードポリシーを取得します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript SDK for Kotlin API リファレンス」の「GetAccountPasswordPolicy」を参照してください。

`GetPolicy`

次の例は、`GetPolicy` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ポリシーを取得します。

```
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);
```

```
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetPolicy」を参照してください。

`GetRole`

次の例は、`GetRole` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ロールを取得します。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetRole」を参照してください。

GetServerCertificate

次の例は、GetServerCertificate を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

サーバー証明書を取得します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetServerCertificate」を参照してください。

GetServiceLinkedRoleDeletionStatus

次の例は、GetServiceLinkedRoleDeletionStatus を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetServiceLinkedRoleDeletionStatus」を参照してください。

`ListAccessKeys`

次の例は、`ListAccessKeys` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アクセスキーを一覧表示します。

```
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;
    }
  }
}
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListAccessKeys」を参照してください。

`ListAccountAliases`

次の例は、`ListAccountAliases` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アカウントエイリアスを一覧表示します。

```
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;
    }
  }
}
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListAccountAliases」を参照してください。

`ListAttachedRolePolicies`

次の例は、`ListAttachedRolePolicies` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ロールにアタッチされたポリシーを一覧表示します。

```
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;
    }
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListAttachedRolePolicies」を参照してください。

ListGroups

次の例は、ListGroups を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

グループを一覧表示します。

```
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;
    }
  }
```

```
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListGroup」を参照してください。

`ListPolicies`

次の例は、`ListPolicies` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ポリシーを一覧表示します。

```
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;
    }
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListPolicies」を参照してください。

`ListRolePolicies`

次の例は、`ListRolePolicies` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ポリシーを一覧表示します。

```
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;
    }
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListRolePolicies」を参照してください。

ListRoles

次の例は、ListRoles を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ロールを一覧表示します。

```
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;
    }
  }
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListRoles」を参照してください。

`ListSAMLProviders`

次の例は、`ListSAMLProviders` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

SAML プロバイダーを一覧表示します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListSAMLProviders」を参照してください。

`ListServerCertificates`

次の例は、`ListServerCertificates` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

証明書を一覧表示します。

```
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;
    }
  }
}
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript SDK for Rust API リファレンス」の「ListServerCertificates」を参照してください。

`ListUsers`

次の例は、`ListUsers` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ユーザーを一覧表示します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListUsers」を参照してください。

PutRolePolicy

次の例は、PutRolePolicy を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutRolePolicy」を参照してください。

`UpdateAccessKey`

次の例は、UpdateAccessKey を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

アクセスキーを更新します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateAccessKey」を参照してください。

`UpdateServerCertificate`

次の例は、`UpdateServerCertificate` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

サーバー証明書を更新します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateServerCertificate」を参照してください。

`UpdateUser`

次の例は、`UpdateUser` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ユーザーを更新します。

```
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);
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateUser」を参照してください。

`UploadServerCertificate`

次の例は、`UploadServerCertificate` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UploadServerCertificate」を参照してください。

シナリオ

レジリエントなサービスの構築と管理

次のコード例は、本、映画、曲のレコメンデーションを返す負荷分散型ウェブサービスの作成方法を示しています。この例は、障害に対するサービスの対応方法と、障害発生時の耐障害性を高めるためにサービスを再構築する方法を示しています。

- Amazon EC2 Auto Scaling グループを使用して、起動テンプレートに基づいて Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを作成し、インスタンス数を所定の範囲内に維持します。
- Elastic Load Balancing で HTTP リクエストを処理して配信します。
- Auto Scaling グループ内のインスタンスの状態を監視し、正常なインスタンスにのみリクエストを転送します。
- 各 EC2 インスタンスで Python ウェブサーバーを実行して HTTP リクエストを処理します。ウェブサーバーはレコメンデーションとヘルスチェックを返します。
- Amazon DynamoDB テーブルを使用してレコメンデーションサービスをシミュレートできます。
- AWS Systems Manager パラメータを更新して、リクエストとヘルスチェックに対するウェブサーバーの応答を制御します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
#!/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.",
  });
```

```
}
```

すべてのリソースをデプロイするための手順を作成します。

```
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,
];
```

デモを実行するための手順を作成します。

```
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;
}
```

すべてのリソースを破棄するための手順を作成します。

```
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.`);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - 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 SDK for JavaScript (v3) を使用した の例

次のコード例は、 で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています AWS IoT SiteWise。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

こんにち AWS IoT SiteWiseは

次のコード例は、 AWS IoT SiteWiseの使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のListAssetModels」を参照してください。

トピック

- 基本
- アクション

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- AWS IoT SiteWise アセットモデルを作成します。
- AWS IoT SiteWise アセットを作成します。
- プロパティ ID 値を取得します。
- AWS IoT SiteWise アセットにデータを送信します。
- Asset AWS IoT SiteWise プロパティの値を取得します。
- AWS IoT SiteWise ポータルを作成します。
- AWS IoT SiteWise ゲートウェイを作成します。
- AWS IoT SiteWise ゲートウェイを記述します。
- AWS IoT SiteWise アセットを削除します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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 });
}
```

アクション

`BatchPutAssetPropertyValue`

次の例は、`BatchPutAssetPropertyValue` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のBatchPutAssetPropertyValue」を参照してください。

`CreateAsset`

次の例は、`CreateAsset` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCreateAsset」を参照してください。

`CreateAssetModel`

次の例は、`CreateAssetModel` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCreateAssetModel」を参照してください。

`CreateGateway`

次の例は、`CreateGateway` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」の「CreateGateway」を参照してください。

`CreatePortal`

次の例は、`CreatePortal` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import {
  CreatePortalCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create a Portal.
 * @param {{ portalName: string, portalContactEmail: string, roleArn: string }}
 */
export const main = async ({ portalName, portalContactEmail, roleArn }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreatePortalCommand({
        portalName: portalName, // The name to give the created Portal.
        portalContactEmail: portalContactEmail, // A valid contact email.
        roleArn: roleArn, // The ARN of a service role that allows the portal's
 users to access the portal's resources.
      }),
    );
    console.log("Portal created successfully.");
    return result;
  } catch (caught) {
```

```
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the Portal.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API CreatePortal」を参照してください。

`DeleteAsset`

次の例は、`DeleteAsset` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeleteAsset」を参照してください。

`DeleteAssetModel`

次の例は、`DeleteAssetModel` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「API リファレンス」のDeleteAssetModel」を参照してください。

`DeleteGateway`

次の例は、`DeleteGateway` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeleteGateway」を参照してください。

`DeletePortal`

次の例は、`DeletePortal` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import {
  DeletePortalCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * List asset models.
 * @param {{ portalId : string }}
 */
export const main = async ({ portalId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeletePortalCommand({
        portalId: portalId, // The id of the portal.
      }),
    );
    console.log("Portal deleted successfully.");
    return { portalDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the portal. Please check
 the portal id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeletePortal」を参照してください。

`DescribeAssetModel`

次の例は、`DescribeAssetModel` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」の「DescribeAssetModel」を参照してください。

`DescribeGateway`

次の例は、`DescribeGateway` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」の「DescribeGateway」を参照してください。

`DescribePortal`

次の例は、`DescribePortal` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import {
  DescribePortalCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe a portal.
 * @param {{ portalId: string }}
 */
export const main = async ({ portalId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new DescribePortalCommand({
        portalId: portalId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Portal information retrieved successfully.");
    return result;
```

```
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Portal could not be found. Please check the Portal
 Id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」の「DescribePortal」を参照してください。

GetAssetPropertyValue

次の例は、GetAssetPropertyValue を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のGetAssetPropertyValue」を参照してください。

`ListAssetModels`

次の例は、`ListAssetModels` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のListAssetModels」を参照してください。

SDK for JavaScript (v3) を使用した Kinesis の例

次のコード例は、Kinesis で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション
- サーバーレスサンプル

アクション

PutRecords

次の例は、PutRecords を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutRecords」を参照してください。

サーバーレスサンプル

Kinesis トリガーから Lambda 関数を呼び出す

次のコード例では、Kinesis ストリームからレコードを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数は Kinesis ペイロードを取得し、それを Base64 からデコードして、そのレコードの内容を口グ記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での Kinesis イベントの消費。

```
// 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;
}
```

TypeScript を使用した Lambda での Kinesis イベントの消費。

```
// 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;
}
```

Kinesis トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、Kinesis ストリームからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Javascript を使用した Lambda での Kinesis バッチアイテム失敗のレポート。

```
// 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;
}
```

TypeScript を使用した Lambda での Kinesis バッチアイテム失敗のレポート。

```
// 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;
}
```

SDK for JavaScript (v3) を使用した Lambda 例

次のコード例は、Lambda で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Lambda

次のコード例では、Lambda の使用を開始する方法について示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListFunctions」を参照してください。

トピック

- 基本
- アクション
- シナリオ
- サーバーレスサンプル

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- IAM ロールと Lambda 関数を作成し、ハンドラーコードをアップロードします。
- 1 つのパラメーターで関数を呼び出して、結果を取得します。
- 関数コードを更新し、環境変数で設定します。
- 新しいパラメーターで関数を呼び出して、結果を取得します。返された実行ログを表示します。
- アカウントの関数を一覧表示し、リソースをクリーンアップします。

詳細については、「コンソールで Lambda 関数を作成する」を参照してください。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ログに書き込むアクセス許可を Lambda に付与する AWS Identity and Access Management (IAM) ロールを作成します。

```
    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);
};
```

Lambda 関数を作成し、ハンドラーコードをアップロードします。

```
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);
};
```

1 つのパラメーターで関数を呼び出して、結果を取得します。

```
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 };
};
```

関数コードを更新し、Lambda 環境を環境可変で設定します。

```
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;
};
```

アカウントの関数を一覧表示します。

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

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

IAM ロールと Lambda 関数を削除します。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

アクション

`CreateFunction`

次の例は、`CreateFunction` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateFunction」を参照してください。

`DeleteFunction`

次の例は、`DeleteFunction` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteFunction」を参照してください。

`GetFunction`

次の例は、`GetFunction` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetFunction」を参照してください。

`Invoke`

次の例は、`Invoke` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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 };
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Invoke」を参照してください。

`ListFunctions`

次の例は、`ListFunctions` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const listFunctions = () => {
```

```
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListFunctions」を参照してください。

`UpdateFunctionCode`

次の例は、`UpdateFunctionCode` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateFunctionCode」を参照してください。

`UpdateFunctionConfiguration`

次の例は、`UpdateFunctionConfiguration` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateFunctionConfiguration」を参照してください。

シナリオ

Lambda 関数を使用して登録済みのユーザーを自動的に確認する

次のコード例は、Lambda 関数を使用して登録済みの Amazon Cognito ユーザーを確認する方法を示しています。

- `PreSignUp` トリガーの Lambda 関数を呼び出すようにユーザープールを設定します。
- Amazon Cognito でユーザーをサインアップする
- Lambda 関数は DynamoDB テーブルをスキャンし、登録済みのユーザーを自動的に確認します。
- 新しいユーザーとしてサインインし、リソースをクリーンアップします。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

インタラクティブな「シナリオ」実行を設定します。JavaScript (v3) の例では、シナリオランナーを共有して、複雑な例を効率化します。完全なソースコードは GitHub にあります。

```
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.",
  });
}
```

このシナリオでは、既知のユーザーを自動確認する方法を示します。サンプルのステップをオーケストレーションします。

```
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,
  );
```

これらは、他のシナリオと共有されるステップです。

```
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 },
);
```

Lambda 関数を使用する PreSignUp トリガーのハンドラー。

```
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);
};
```

CloudWatch Logs アクションのモジュール。

```
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];
  }
};
```

Amazon Cognito アクションのモジュール。

```
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];
  }
};
```

DynamoDB アクションのモジュール。

```
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];
  }
};
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

顧客からのフィードバックを分析するアプリケーションの作成

次のコード例は、顧客のコメントカードを分析し、元の言語から翻訳し、顧客の感情を判断し、翻訳されたテキストから音声ファイルを生成するアプリケーションの作成方法を示しています。

SDK for JavaScript (v3)

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、GitHub のプロジェクトを参照してください。次の抜粋 AWS SDK for JavaScript は、Lambda 関数内で がどのように使用されるかを示しています。

```
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 };
};
```

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

ブラウザからの Lambda 関数の呼び出し

次のコード例は、ブラウザから AWS Lambda 関数を呼び出す方法を示しています。

SDK for JavaScript (v3)

AWS Lambda 関数を使用して Amazon DynamoDB テーブルをユーザー選択で更新するブラウザベースのアプリケーションを作成できます。このアプリは v3 AWS SDK for JavaScript を使用します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Lambda

API Gateway を使用して Lambda 関数を呼び出す

次のコード例は、Amazon API Gateway によって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

Lambda JavaScript ランタイム API を使用して AWS Lambda 関数を作成する方法を示します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、Amazon API Gateway によって呼び出される Lambda 関数を作成する方法を示します。

この関数は、Amazon DynamoDB テーブルをスキャンして、Amazon Simple Notification Service (Amazon SNS) を使用して、従業員に年間の記念日を祝福するテキストメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

スケジュールされたイベントを使用した Lambda 関数の呼び出し

次のコード例は、Amazon EventBridge スケジュールされたイベントによって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge

- Lambda
- Amazon SNS

サーバーレスサンプル

Lambda 関数での Amazon RDS データベースへの接続

次のコード例は、RDS データベースに接続する Lambda 関数を実装する方法を示しています。この関数は、シンプルなデータベースリクエストを実行し、結果を返します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Javascript を使用した Lambda 関数での Amazon RDS データベースへの接続

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

TypeScript を使用した Lambda 関数での Amazon RDS データベースへの接続

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

Kinesis トリガーから Lambda 関数を呼び出す

次のコード例では、Kinesis ストリームからレコードを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数は Kinesis ペイロードを取得し、それを Base64 からデコードして、そのレコードの内容を ログ記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での Kinesis イベントの消費。

```
// 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;
}
```

TypeScript を使用した Lambda での Kinesis イベントの消費。

```
// 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;
}
```

DynamoDB トリガーから Lambda 関数を呼び出す

次のコード例は、DynamoDB ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DynamoDB ペイロードを取得し、レコードの内容を口グ記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で DynamoDB イベントの消費。

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

TypeScript を使用した Lambda での DynamoDB イベントの消費。

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

Amazon DocumentDB トリガーから Lambda 関数を呼び出す

次のコード例は、DocumentDB 変更ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DocumentDB ペイロードを取得し、レコードの内容をログ記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で Amazon DocumentDB イベントの消費。

```
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));
};
```

TypeScript を使用して Lambda で Amazon DocumentDB イベントの消費。

```
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));
};
```

Amazon MSK トリガーから Lambda 関数を呼び出す

次のコード例は、Amazon MSK クラスターからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は MSK ペイロードを取得し、レコードの内容をログ記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での Amazon MSK イベントの消費。

```
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)
      })
    }
}
```

TypeScript を使用した Lambda での Amazon MSK イベントの消費。

```
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;
      }
    };
  }
}
```

Amazon S3 トリガーから Lambda 関数を呼び出す

次のコード例は、S3 バケットにオブジェクトをアップロードすることによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数は、イベントパラメータから S3 バケット名とオブジェクトキーを取得し、Amazon S3 API を呼び出してオブジェクトのコンテンツタイプを取得してログに記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で S3 イベントを消費します。

```
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);
    }
};
```

TypeScript を使用して Lambda で S3 イベントを消費する。

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

Amazon SNS トリガーから Lambda 関数を呼び出す

次のコード例は、SNS トピックからメッセージを受信することによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行する方法を確認してください。

JavaScript を使用した Lambda での SNS イベントの消費。

```
// 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;
  }
}
```

TypeScript を使用した Lambda での SNS イベントの消費。

```
// 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;
  }
}
```

Amazon SQS トリガーから Lambda 関数を呼び出す

次のコード例では、SQS キューからメッセージを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での SQS イベントの消費。

```
// 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;
  }
}
```

TypeScript を使用した Lambda での SQS イベントの消費。

```
// 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;
  }
}
```

Kinesis トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、Kinesis ストリームからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Javascript を使用した Lambda での Kinesis バッチアイテム失敗のレポート。

```
// 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;
}
```

TypeScript を使用した Lambda での Kinesis バッチアイテム失敗のレポート。

```
// 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;
}
```

DynamoDB トリガーで Lambda 関数のバッチアイテムの失敗をレポートする

次のコード例は、DynamoDB ストリームからイベントを受信する Lambda 関数に部分的なバッチレスポンスを実装する方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
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: [] };
};
```

TypeScript を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
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 };
};
```

Amazon SQS トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、SQS キューからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での SQS バッチアイテム失敗のレポート。

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

TypeScript を使用して Lambda で SQS バッチ項目の失敗を報告します。

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

SDK for JavaScript (v3) を使用した Amazon Lex の例

次のコード例は、Amazon Lex で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- シナリオ

シナリオ

Amazon Lex chatbot を構築する

次のコード例は、ウェブサイトの訪問者を引き付けるチャットボットを作成する方法を示しています。

SDK for JavaScript (v3)

ウェブアプリケーション内に Amazon Lex chatbotを作成して、ウェブサイトの訪問者に対応することができます。

完全なソースコードとセットアップと実行の手順については、 デ AWS SDK for JavaScript ベロッパーガイドのAmazon Lexチャットボットの構築」の完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

SDK for JavaScript (v3) を使用した Amazon Location の例

次のコード例は、Amazon Location で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

こんにちは Amazon Location Service

次のコード例は、Amazon Location Service の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のListGeofencesPaginator」を参照してください。

トピック

- 基本

- アクション

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- Amazon Location マップを作成します。
- Amazon Location API キーを作成します。
- マップ URL を表示します。
- ジオフェンスコレクションを作成します。
- ジオフェンスジオメトリを保存します。
- トラッカーリソースを作成します。
- デバイスの位置を更新します。
- 指定されたデバイスの最新の位置更新を取得します。
- ルート計算ツールを作成します。
- シアトルとバンクーバー間の距離を決定します。
- Amazon Location の上位レベルの APIs。
- Amazon Location Assets を削除します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/*
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 });
}
```

アクション

BatchUpdateDevicePosition

次の例は、BatchUpdateDevicePosition を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のBatchUpdateDevicePosition」を参照してください。

`CalculateRoute`

次の例は、`CalculateRoute` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCalculateRoute」を参照してください。

`CreateGeofenceCollection`

次の例は、`CreateGeofenceCollection` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」の「CreateGeofenceCollection」を参照してください。

`CreateMap`

次の例は、`CreateMap` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCreateMap」を参照してください。

`CreateRouteCalculator`

次の例は、`CreateRouteCalculator` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCreateRouteCalculator」を参照してください。

`CreateTracker`

次の例は、`CreateTracker` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCreateTracker」を参照してください。

`DeleteGeofenceCollection`

次の例は、`DeleteGeofenceCollection` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeleteGeofenceCollection」を参照してください。

`DeleteMap`

次の例は、`DeleteMap` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeleteMap」を参照してください。

`DeleteRouteCalculator`

次の例は、`DeleteRouteCalculator` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeleteRouteCalculator」を参照してください。

`DeleteTracker`

次の例は、`DeleteTracker` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のDeleteTracker」を参照してください。

`GetDevicePosition`

次の例は、`GetDevicePosition` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」の[GetDevicePosition」を参照してください。

`PutGeofence`

次の例は、`PutGeofence` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。[AWS コード例リポジトリ] で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」の「PutGeofence」を参照してください。

SDK for JavaScript (v3) を使用した Amazon MSK の例

次のコード例は、Amazon MSK で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

サーバーレスサンプル

Amazon MSK トリガーから Lambda 関数を呼び出す

次のコード例は、Amazon MSK クラスターからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は MSK ペイロードを取得し、レコードの内容をログ記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での Amazon MSK イベントの消費。

```
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)
      })
    }
}
```

TypeScript を使用した Lambda での Amazon MSK イベントの消費。

```
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;
      }
    };
  }
}
```

SDK for JavaScript (v3) を使用した Amazon Personalize の例

次のコード例は、Amazon Personalize で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

アクション

`CreateBatchInferenceJob`

次の例は、`CreateBatchInferenceJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateBatchInferenceJob」を参照してください。

CreateBatchSegmentJob

次の例は、CreateBatchSegmentJob を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateBatchSegmentJob」を参照してください。

`CreateCampaign`

次の例は、`CreateCampaign` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateCampaign」を参照してください。

`CreateDataset`

次の例は、`CreateDataset` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateDataset」を参照してください。

`CreateDatasetExportJob`

次の例は、`CreateDatasetExportJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateDatasetExportJob」を参照してください。

`CreateDatasetGroup`

次の例は、`CreateDatasetGroup` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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);
```

ドメインデータセットグループを作成します。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateDatasetGroup」を参照してください。

`CreateDatasetImportJob`

次の例は、`CreateDatasetImportJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateDatasetImportJob」を参照してください。

`CreateEventTracker`

次の例は、`CreateEventTracker` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateEventTracker」を参照してください。

`CreateFilter`

次の例は、`CreateFilter` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateFilter」を参照してください。

`CreateRecommender`

次の例は、`CreateRecommender` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateRecommender」を参照してください。

`CreateSchema`

次の例は、`CreateSchema` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

ドメインでスキーマを作成します。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateSchema」を参照してください。

CreateSolution

次の例は、CreateSolution を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateSolution」を参照してください。

`CreateSolutionVersion`

次の例は、`CreateSolutionVersion` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateSolutionVersion」を参照してください。

SDK for JavaScript (v3) を使用した Amazon Personalize Events の例

次のコード例は、Amazon Personalize Events で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

アクション

`PutEvents`

次の例は、`PutEvents` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutEvents」を参照してください。

PutItems

次の例は、PutItems を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutItems」を参照してください。

PutUsers

次の例は、PutUsers を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutUsers」を参照してください。

SDK for JavaScript (v3) による Amazon Personalize Runtime の例

次のコード例は、Amazon Personalize ランタイムで AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`GetPersonalizedRanking`

次の例は、`GetPersonalizedRanking` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetPersonalizedRanking」を参照してください。

`GetRecommendations`

次の例は、`GetRecommendations` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// 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();
```

フィルターを使用してレコメンデーションを取得します (カスタムデータセットグループ)。

```
// 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();
```

ドメインデータセットグループで作成されたレコメンダーから、フィルタリングされたレコメンデーションを取得します。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetRecommendations」を参照してください。

SDK for JavaScript (v3) を使用した Amazon Pinpoint の例

次のコード例は、Amazon Pinpoint で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`SendMessages`

次の例は、`SendMessages` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
export const pinClient = new PinpointClient({ region: REGION });
```

E メールメッセージを送信します。

```
// 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();
```

SMS メッセージを送信します。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SendMessage」を参照してください。

SDK for JavaScript (v3) を使用した Amazon Polly の例

次のコード例は、Amazon Polly で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- シナリオ

シナリオ

顧客からのフィードバックを分析するアプリケーションの作成

次のコード例は、顧客のコメントカードを分析し、元の言語から翻訳し、顧客の感情を判断し、翻訳されたテキストから音声ファイルを生成するアプリケーションの作成方法を示しています。

SDK for JavaScript (v3)

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、GitHub のプロジェクトを参照してください。次の抜粋 AWS SDK for JavaScript は、Lambda 関数内で がどのように使用されるかを示しています。

```
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 };
};
```

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

SDK for JavaScript (v3) を使用した Amazon RDS の例

次のコード例は、Amazon RDS で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

シナリオ

Aurora Serverless 作業項目トラッカーの作成

次のコード例は、Amazon Aurora Serverless データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを送信するウェブアプリケーションを作成する方法を示しています。

SDK for JavaScript (v3)

AWS SDK for JavaScript (v3) を使用して Amazon Aurora データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信するウェブアプリケーションを作成する方法を示します。この例では、React.js で構築されたフロントエンドを使用して Express Node.js バックエンドと対話します。

- React.js ウェブアプリケーションを と統合します AWS のサービス。
- Aurora テーブルの項目を一覧表示、追加、更新します。
- Amazon SES を使用して、フィルター処理された作業項目の E メールレポートを送信します。
- 含まれている AWS CloudFormation スクリプトを使用してサンプルリソースをデプロイおよび管理します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス
- Amazon SES

サーバーレスサンプル

Lambda 関数での Amazon RDS データベースへの接続

次のコード例は、RDS データベースに接続する Lambda 関数を実装する方法を示しています。この関数は、シンプルなデータベースリクエストを実行し、結果を返します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Javascript を使用した Lambda 関数での Amazon RDS データベースへの接続

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

TypeScript を使用した Lambda 関数での Amazon RDS データベースへの接続

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

SDK for JavaScript (v3) を使用した Amazon RDS Data Service の例

次のコード例は、Amazon RDS Data Service で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- シナリオ

シナリオ

Aurora Serverless 作業項目トラッカーの作成

次のコード例は、Amazon Aurora Serverless データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを送信するウェブアプリケーションを作成する方法を示しています。

SDK for JavaScript (v3)

AWS SDK for JavaScript (v3) を使用して Amazon Aurora データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信するウェブアプリケーションを作成する方法を示します。この例では、React.js で構築されたフロントエンドを使用して Express Node.js バックエンドと対話します。

- React.js ウェブアプリケーションを と統合します AWS のサービス。
- Aurora テーブルの項目を一覧表示、追加、更新します。
- Amazon SES を使用して、フィルター処理された作業項目の E メールレポートを送信します。
- 含まれている AWS CloudFormation スクリプトを使用してサンプルリソースをデプロイおよび管理します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス
- Amazon SES

SDK for JavaScript (v3) を使用した Amazon Redshift の例

次のコード例は、Amazon Redshift で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

アクション

`CreateCluster`

次の例は、`CreateCluster` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成します。

```
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 };
```

クラスターを作成します。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateCluster」を参照してください。

`DeleteCluster`

次の例は、`DeleteCluster` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成します。

```
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 };
```

クラスターを作成します。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteCluster」を参照してください。

DescribeClusters

次の例は、DescribeClusters を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成します。

```
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 };
```

クラスターを記述します。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeClusters」を参照してください。

`ModifyCluster`

次の例は、`ModifyCluster` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成します。

```
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 };
```

クラスターを変更します。

```
// 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();
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ModifyCluster」を参照してください。

SDK for JavaScript (v3) を使用した Amazon Rekognition の例 JavaScript

次のコード例は、Amazon Rekognition で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

シナリオ

サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

イメージ内のオブジェクトを検出する

次のコード例は、Amazon Rekognition を使用して画像内のオブジェクトをカテゴリ別に検出するアプリを構築する方法を示しています。

SDK for JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Rekognition を使用して Amazon Simple Storage Service (Amazon S3) バケットにあるイメージ内のオブジェクトをカテゴリ別に識別するアプリケーション AWS SDK for JavaScript を作成する方法を示します。アプリケーションは Amazon Simple Email Service (Amazon SES) を使用して、結果を含む E メール通知を管理者に送信します。

以下ではその方法を説明しています。

- Amazon Cognito を使用して認証されていないユーザーを作成します。
- Amazon Rekognition を使用して、オブジェクトのイメージを分析します。
- Amazon SES の E メールアドレスを検証します。
- Amazon SES を使用して、E メール通知を送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

SDK for JavaScript (v3) を使用した Amazon S3 の例

次のコード例は、Amazon S3 で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Amazon S3

次のコード例は、Amazon S3 の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListBuckets」を参照してください。

トピック

- 基本
- アクション
- シナリオ
- サーバーレスサンプル

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- バケットを作成し、そこにファイルをアップロードします。
- バケットからオブジェクトをダウンロードします。
- バケット内のサブフォルダにオブジェクトをコピーします。
- バケット内のオブジェクトを一覧表示します。
- バケットオブジェクトとバケットを削除します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

まず、必要なモジュールをすべてインポートします。

```
// 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";
```

前述のインポートでは、いくつかのヘルパーユーティリティを参照しています。これらのユーティリティは、このセクションの冒頭でリンクされている GitHub リポジトリのローカルにあります。参考までに、これらのユーティリティの以下の実装を参照してください。

```
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`;
};
```

S3 のオブジェクトは「バケット」に保存されます。新しいバケットを作成する関数を定義しましょう。

```
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;
};
```

バケットには「オブジェクト」が含まれています。この関数は、ディレクトリの内容をバケットにオブジェクトとしてアップロードします。

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

オブジェクトをアップロードしたら、正しくアップロードされたことを確認します。そのためには ListObjects を使用できます。ここでは 'Key' プロパティを使用しますが、レスポンスには他にも便利なプロパティがあります。

```
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`);
};
```

バケットから別のバケットにオブジェクトをコピーしたい場合があります。そのためには、CopyObject コマンドを使用してください。

```
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();
};
```

バケットから複数のオブジェクトを取得するための SDK メソッドはありません。代わりに、ダウンロードして繰り返し処理するオブジェクトのリストを作成します。

```
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");
};
```

では、リソースをクリーンアップしましょう。バケットを削除するには、そのバケットを空にしておく必要があります。これら 2 つの関数はバケットを空にして削除します。

```
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`);
};
```

「main」関数はすべてをまとめます。このファイルを直接実行すると、main 関数が呼び出されます。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。

- CopyObject
- CreateBucket
- DeleteBucket
- DeleteObjects
- GetObject
- ListObjectsV2
- PutObject

アクション

`CopyObject`

次の例は、`CopyObject` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

オブジェクトをコピーします。

```
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;
    }
  }
};
```

ETag が指定されたオブジェクトと一致しないことを条件にオブジェクトをコピーします。

```
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"));
  }
```

```
}
```

ETag が指定されたオブジェクトと一致しないことを条件にオブジェクトをコピーします。

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

特定の期間に作成または変更された条件に基づいて、 を使用してオブジェクトをコピーします。

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

特定の期間に作成または変更されていないことを条件に、 を使用してオブジェクトをコピーします。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CopyObject」を参照してください。

CreateBucket

次の例は、CreateBucket を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケットを作成します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateBucket」を参照してください。

`DeleteBucket`

次の例は、`DeleteBucket` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケットを削除します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteBucket」を参照してください。

DeleteBucketPolicy

次の例は、DeleteBucketPolicy を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケット ポリシーを削除します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteBucketPolicy」を参照してください。

`DeleteBucketWebsite`

次の例は、`DeleteBucketWebsite` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケットからウェブサイト設定を削除します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteBucketWebsite」を参照してください。

DeleteObject

次の例は、DeleteObject を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

オブジェクトを削除します。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteObject」を参照してください。

`DeleteObjects`

次の例は、`DeleteObjects` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

複数のオブジェクトを削除します。

```
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;
```

```
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteObjects」を参照してください。

`GetBucketAcl`

次の例は、`GetBucketAcl` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ACL 許可を取得します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetBucketAcl」を参照してください。

`GetBucketCors`

次の例は、`GetBucketCors` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケットの CORS ポリシーを取得します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetBucketCors」を参照してください。

`GetBucketPolicy`

次の例は、`GetBucketPolicy` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケットポリシーを取得します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetBucketPolicy」を参照してください。

`GetBucketWebsite`

次の例は、`GetBucketWebsite` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ウェブサイト設定を取得します。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetBucketWebsite」を参照してください。

`GetObject`

次の例は、`GetObject` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

オブジェクトをダウンロードします。

```
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;
    }
  }
};
```

ETag が指定されたものと一致することを条件に、 オブジェクトをダウンロードします。

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

ETag が指定されたオブジェクトと一致しないことを条件として、 オブジェクトをダウンロードします。

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

特定の期間に作成または変更されたことを条件に、 を使用してオブジェクトをダウンロードします。

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

特定の期間に作成または変更されていないことを条件に、 を使用してオブジェクトをダウンロードします。

```
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"));
  }
```

```
}
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetObject」を参照してください。

`GetObjectLegalHold`

次の例は、`GetObjectLegalHold` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript API リファレンスの「GetObjectLegalHold」を参照してください。

`GetObjectLockConfiguration`

次の例は、`GetObjectLockConfiguration` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetObjectLockConfiguration」を参照してください。

`GetObjectRetention`

次の例は、`GetObjectRetention` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript API リファレンスの「GetObjectRetention」を参照してください。

`ListBuckets`

次の例は、`ListBuckets` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケットを一覧表示します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListBuckets」を参照してください。

`ListObjectsV2`

次の例は、`ListObjectsV2` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケット内のすべてのオブジェクトを一覧表示します。オブジェクトが複数ある場合は、IsTruncated と NextContinuationToken を使用してリスト全体を繰り返し処理します。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListObjectsV2」を参照してください。

`PutBucketAcl`

次の例は、`PutBucketAcl` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケット ACL をプットします。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutBucketAcl」を参照してください。

PutBucketCors

次の例は、PutBucketCors を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CORS ルールを追加します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutBucketCors」を参照してください。

PutBucketPolicy

次の例は、PutBucketPolicy を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ポリシーを追加します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutBucketPolicy」を参照してください。

PutBucketWebsite

次の例は、PutBucketWebsite を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ウェブサイト設定を設定します。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutBucketWebsite」を参照してください。

`PutObject`

次の例は、`PutObject` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

オブジェクトをアップロードします。

```
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;
    }
```

```
  }
};
```

ETag が指定されたものと一致することを条件にオブジェクトをアップロードします。

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

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutObject」を参照してください。

`PutObjectLegalHold`

次の例は、`PutObjectLegalHold` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutObjectLegalHold」を参照してください。

`PutObjectLockConfiguration`

次の例は、`PutObjectLockConfiguration` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケットのオブジェクトロック設定を指定します。

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

バケットのデフォルトの保存期間を設定します。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutObjectLockConfiguration」を参照してください。

`PutObjectRetention`

次の例は、`PutObjectRetention` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutObjectRetention」を参照してください。

シナリオ

署名付き URL を作成する

次のコード例は、Amazon S3 の署名付き URL を作成し、オブジェクトをアップロードする方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

署名付き URL を作成して、オブジェクトをバケットにアップロードします。

```
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;
    }
  }
};
```

署名付き URL を作成して、オブジェクトをバケットからダウンロードします。

```
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;
    }
  }
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。

サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Amazon S3 オブジェクトを一覧表示するウェブページの作成

次のコード例は、ウェブページに Amazon S3 オブジェクトを一覧表示する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

次のコードは、 AWS SDK を呼び出す関連する React コンポーネントです。このコンポーネントを含むアプリケーションの実行可能なバージョンは、前述の GitHub リンクにあります。

```
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;
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListObjects」を参照してください。

Amazon Textract エクスプローラーアプリケーションを作成する

次のコード例は、インタラクティブアプリケーションを使用して Amazon Textract 出力を調べる方法を示しています。

SDK for JavaScript (v3)

を使用して、Amazon Textract を使用してドキュメントイメージからデータを抽出し、インタラクティブなウェブページに表示する React アプリケーション AWS SDK for JavaScript を構築する方法を示します。この例はウェブブラウザで実行され、認証情報に認証された Amazon Cognito ID が必要です。Amazon Simple Storage Service (Amazon S3) をストレージに使用し、通知のために、Amazon Simple Notification Service (Amazon SNS) トピックにサブスクライブした Amazon Simple Queue Service (Amazon SQS) キューをポーリングします。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3

- Amazon SNS
- Amazon SQS
- Amazon Textract

バケット内のすべてのオブジェクトを削除する

次のコード例は、Amazon S3 バケットからすべてのオブジェクトを削除する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon S3 バケットからすべてのオブジェクトを削除します。

```
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);
}
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - DeleteObjects
  - ListObjectsV2

イメージ内のオブジェクトを検出する

次のコード例は、Amazon Rekognition を使用して画像内のオブジェクトをカテゴリ別に検出するアプリを構築する方法を示しています。

SDK for JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Rekognition を使用して Amazon Simple Storage Service (Amazon S3) バケットにあるイメージ内のオブジェクトをカテゴリ別に識別するアプリケーション AWS SDK for JavaScript を作成する方法を示します。アプリケーションは Amazon Simple Email Service (Amazon SES) を使用して、結果を含む E メール通知を管理者に送信します。

以下ではその方法を説明しています。

- Amazon Cognito を使用して認証されていないユーザーを作成します。
- Amazon Rekognition を使用して、オブジェクトのイメージを分析します。
- Amazon SES の E メールアドレスを検証します。
- Amazon SES を使用して、E メール通知を送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

Amazon S3 オブジェクトをロックする

次のコード例は、S3 オブジェクトロック機能を操作する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

シナリオのエントリポイント (index.js) 。これにより、すべての
ステップがオーケストレーションされます。GitHub にアクセスし
て、Scenario、ScenarioInput、ScenarioOutput、ScenarioAction の実装の詳細を確認します。

```
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]",
  });
}
```

コンソール () にウェルカムメッセージを出力します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 };
```

バケット、オブジェクト、およびファイル設定をデプロイします (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,
};
```

バケット内のファイルを表示および削除します (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 };
```

作成されたすべてのリソースを破棄します (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 };
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold
  - PutObjectLockConfiguration
  - PutObjectRetention

条件付きリクエストを行う

次のコード例は、Amazon S3 リクエストに前提条件を追加する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ワークフローのエントリポイント (index.js) 。これにより、すべてのステップがオーケストレーションされます。GitHub にアクセスして、Scenario、ScenarioInput、ScenarioOutput、ScenarioAction の実装の詳細を確認します。

```
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]",
  });
}
```

コンソール () にウェルカムメッセージを出力します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 };
```

バケットとオブジェクトをデプロイします (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,
};
```

S3 条件付きリクエスト () を使用してオブジェクトを取得、コピー、配置します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 };
```

作成されたすべてのリソースを破棄します (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 };
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - CopyObject
  - GetObject
  - PutObject

大きなファイルをアップロードまたはダウンロードする

次のコード例は、Amazon S3 との間で大きなファイルをアップロードまたはダウンロードする方法を示しています。

詳細については、「マルチパートアップロードを使用したオブジェクトのアップロード」を参照してください。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

サイズの大きいファイルをアップロードします。

```
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;
    }
  }
};
```

サイズの大きいファイルをダウンロードします。

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

サーバーレスサンプル

Amazon S3 トリガーから Lambda 関数を呼び出す

次のコード例は、S3 バケットにオブジェクトをアップロードすることによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数は、イベントパラメータから S3 バケット名とオブジェクトキーを取得し、Amazon S3 API を呼び出してオブジェクトのコンテンツタイプを取得してログに記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で S3 イベントを消費します。

```
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);
    }
};
```

TypeScript を使用して Lambda で S3 イベントを消費する。

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

SDK for JavaScript (v3) を使用した S3 Glacier の例

次のコード例は、S3 Glacier で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

アクション

`CreateVault`

次の例は、`CreateVault` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成する

```
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 };
```

ボールトを作成します。

```
// 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();
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスのCreateVaultを参照してください。

`UploadArchive`

次の例は、`UploadArchive` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントの作成

```
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 };
```

アーカイブのアップロード

```
// 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();
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UploadArchive」を参照してください。

SDK for JavaScript (v3) を使用した SageMaker AI の例

次のコード例は、SageMaker AI で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello SageMaker AI

次のコード例は、SageMaker AI の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListNotebookInstances」を参照してください。

トピック

- アクション
- シナリオ

アクション

`CreatePipeline`

次の例は、CreatePipeline を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ローカルで提供される JSON 定義を使用して SageMaker AI パイプラインを作成する関数。

```
/**
 * 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 }),
      );
    },
  };
}
```

- API の詳細については、AWS SDK for JavaScript 「 API リファレンス」のCreatePipeline」を参照してください。

`DeletePipeline`

次の例は、`DeletePipeline` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

SageMaker AI パイプラインを削除するための構文。このコードは、より大きな関数の一部です。コンテキストの詳細については、「パイプラインの作成」または GitHub リポジトリを参照してください。

```
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeletePipeline」を参照してください。

`DescribePipelineExecution`

次の例は、`DescribePipelineExecution` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

SageMaker AI パイプラインの実行が成功、失敗、または停止するのを待ちます。

```
/**
 * 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);
}
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「DescribePipelineExecution」を参照してください。

StartPipelineExecution

次の例は、StartPipelineExecution を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

SageMaker AI パイプラインの実行を開始します。

```
/**
 * 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,
  };
}
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「StartPipelineExecution」を参照してください。

シナリオ

地理空間ジョブとパイプラインの使用を開始する

次のコードサンプルは、以下の操作方法を示しています。

- パイプラインのリソースを設定します。
- 地理空間ジョブを実行するパイプラインを設定します。
- パイプラインの実行を開始します。
- ジョブ実行のステータスをモニタリングします。
- パイプラインの出力を表示します。
- リソースをクリーンアップします。

詳細については、「Community.aws で SDK を使用して SageMaker パイプラインを作成および実行する AWS SDKs」を参照してください。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

次のファイルの抜粋には、SageMaker AI クライアントを使用してパイプラインを管理する関数が含まれています。

```
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();
}
```

この関数は、前述のライブラリ関数を使用して SageMaker AI パイプラインをセットアップし、実行し、作成されたすべてのリソースを削除するファイルからの抜粋です。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution
  - UpdatePipeline

SDK for JavaScript (v3) を使用した Secrets Manager の例

次のコード例は、Secrets Manager で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック
- アクション

アクション

`GetSecretValue`

次の例は、`GetSecretValue` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetSecretValue」を参照してください。

SDK for JavaScript (v3) を使用した Amazon SES の例

次のコード例は、Amazon SES で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

アクション

`CreateReceiptFilter`

次の例は、`CreateReceiptFilter` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
```

```
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateReceiptFilter」を参照してください。

`CreateReceiptRule`

次の例は、`CreateReceiptRule` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateReceiptRule」を参照してください。

`CreateReceiptRuleSet`

次の例は、`CreateReceiptRuleSet` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateReceiptRuleSet」を参照してください。

`CreateTemplate`

次の例は、`CreateTemplate` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateTemplate」を参照してください。

`DeleteIdentity`

次の例は、`DeleteIdentity` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
```

```
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteIdentity」を参照してください。

`DeleteReceiptFilter`

次の例は、`DeleteReceiptFilter` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteReceiptFilter」を参照してください。

DeleteReceiptRule

次の例は、DeleteReceiptRule を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteReceiptRule」を参照してください。

`DeleteReceiptRuleSet`

次の例は、`DeleteReceiptRuleSet` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteReceiptRuleSet」を参照してください。

DeleteTemplate

次の例は、DeleteTemplate を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteTemplate」を参照してください。

GetTemplate

次の例は、GetTemplate を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetTemplate」を参照してください。

`ListIdentities`

次の例は、`ListIdentities` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListIdentities」を参照してください。

`ListReceiptFilters`

次の例は、`ListReceiptFilters` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListReceiptFilters」を参照してください。

`ListTemplates`

次の例は、`ListTemplates` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
```

```
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListTemplates」を参照してください。

`SendBulkTemplatedEmail`

次の例は、`SendBulkTemplatedEmail` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SendBulkTemplatedEmail」を参照してください。

SendEmail

次の例は、SendEmail を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SendEmail」を参照してください。

SendRawEmail

次の例は、SendRawEmail を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

nodemailer を使用して、添付ファイル付きの E メールを送信します。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SendRawEmail」を参照してください。

`SendTemplatedEmail`

次の例は、`SendTemplatedEmail` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SendTemplatedEmail」を参照してください。

`UpdateTemplate`

次の例は、`UpdateTemplate` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateTemplate」を参照してください。

`VerifyDomainIdentity`

次の例は、`VerifyDomainIdentity` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「VerifyDomainIdentity」を参照してください。

`VerifyEmailIdentity`

次の例は、`VerifyEmailIdentity` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「VerifyEmailIdentity」を参照してください。

シナリオ

Amazon Transcribe ストリーミングアプリケーションを構築する

次のコード例は、ライブ音声をリアルタイムで記録、転写、翻訳し、結果を E メールで送信するアプリケーションを構築する方法を示しています。

SDK for JavaScript (v3)

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、転写、翻訳し、Amazon Simple Email Service (Amazon SES) を使用して結果を E メールで送信するアプリケーションを構築する方法について説明します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Aurora Serverless 作業項目トラッカーの作成

次のコード例は、Amazon Aurora Serverless データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを送信するウェブアプリケーションを作成する方法を示しています。

SDK for JavaScript (v3)

AWS SDK for JavaScript (v3) を使用して Amazon Aurora データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信するウェブアプリケーションを作成する方法を示します。この例では、React.js で構築されたフロントエンドを使用して Express Node.js バックエンドと対話します。

- React.js ウェブアプリケーションを と統合します AWS のサービス。
- Aurora テーブルの項目を一覧表示、追加、更新します。
- Amazon SES を使用して、フィルター処理された作業項目の E メールレポートを送信します。
- 含まれている AWS CloudFormation スクリプトを使用してサンプルリソースをデプロイおよび管理します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス
- Amazon SES

イメージ内のオブジェクトを検出する

次のコード例は、Amazon Rekognition を使用して画像内のオブジェクトをカテゴリ別に検出するアプリを構築する方法を示しています。

SDK for JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Rekognition を使用して Amazon Simple Storage Service (Amazon S3) バケットにあるイメージ内のオブジェクトをカテゴリ別に識別するアプリケーション AWS SDK for JavaScript を作成する方法を示します。アプリケーションは Amazon Simple Email Service (Amazon SES) を使用して、結果を含む E メール通知を管理者に送信します。

以下ではその方法を説明しています。

- Amazon Cognito を使用して認証されていないユーザーを作成します。
- Amazon Rekognition を使用して、オブジェクトのイメージを分析します。

- Amazon SES の E メールアドレスを検証します。
- Amazon SES を使用して、E メール通知を送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

SDK for JavaScript (v3) を使用した Amazon SNS の例

次のコード例は、Amazon SNS で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Amazon SNS

以下のコード例は、Amazon SNS の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

SNS クライアントを初期化し、アカウントのトピックを一覧表示します。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListTopics」を参照してください。

トピック

- アクション
- シナリオ
- サーバーレスサンプル

アクション

`CheckIfPhoneNumberIsOptedOut`

次の例は、`CheckIfPhoneNumberIsOptedOut` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの CheckIfPhoneNumberIsOptedOut を参照してください。

`ConfirmSubscription`

次の例は、`ConfirmSubscription` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ConfirmSubscription」を参照してください。

`CreateTopic`

次の例は、`CreateTopic` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateTopic」を参照してください。

`DeleteTopic`

次の例は、`DeleteTopic` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

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

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteTopic」を参照してください。

`GetSMSAttributes`

次の例は、`GetSMSAttributes` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「GetSMSAttributes」を参照してください。

GetTopicAttributes

次の例は、GetTopicAttributes を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

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

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「GetTopicAttributes」を参照してください。

`ListSubscriptions`

次の例は、`ListSubscriptions` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。

- API の詳細については、AWS SDK for JavaScript API リファレンスの「ListSubscriptions」を参照してください。

`ListTopics`

次の例は、`ListTopics` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「ListTopics」を参照してください。

`Publish`

次の例は、`Publish` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

グループ、重複、属性のオプションを指定してメッセージをトピックに発行します。

```
  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();
    }
  }
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「発行」を参照してください。

SetSMSAttributes

次の例は、SetSMSAttributes を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「SetSMSAttributes」を参照してください。

SetTopicAttributes

次の例は、SetTopicAttributes を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SetTopicAttributes」を参照してください。

`Subscribe`

次の例は、`Subscribe` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

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

モバイルアプリケーションをトピックにサブスクライブします。

```
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;
};
```

Lambda 関数をトピックにサブスクライブします。

```
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;
};
```

SQS キューをトピックにサブスクライブします。

```
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;
};
```

トピックにフィルターでサブスクライブします。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「Subscribe」を参照してください。

`Unsubscribe`

次の例は、Unsubscribe を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

別のモジュールでクライアントを作成し、エクスポートします。

```
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({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
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;
};
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「Unsubscribe」を参照してください。

シナリオ

DynamoDB テーブルにデータを送信するアプリケーションを構築する

次のコード例は、Amazon DynamoDB テーブルにデータを送信し、ユーザーがテーブルを更新したときに通知するアプリケーションを構築する方法を示しています。

SDK for JavaScript (v3)

この例では、ユーザーが Amazon DynamoDB テーブルにデータを送信し、Amazon Simple Notification Service (Amazon SNS) を使用して管理者にテキストメッセージを送信できるようにするアプリを構築する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- DynamoDB
- Amazon SNS

サーバーレスアプリケーションを作成して写真の管理

次のコード例では、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションを作成する方法について示しています。

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Amazon Textract エクスプローラーアプリケーションを作成する

次のコード例は、インタラクティブアプリケーションを使用して Amazon Textract 出力を調べる方法を示しています。

SDK for JavaScript (v3)

を使用して、Amazon Textract を使用してドキュメントイメージからデータを抽出し、インタラクティブなウェブページに表示する React アプリケーション AWS SDK for JavaScript を構築する方法を示します。この例はウェブブラウザで実行され、認証情報に認証された Amazon Cognito ID が必要です。Amazon Simple Storage Service (Amazon S3) をストレージに使用し、通知のために、Amazon Simple Notification Service (Amazon SNS) トピックにサブスクライブした Amazon Simple Queue Service (Amazon SQS) キューをポーリングします。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

メッセージをキューに発行する

次のコードサンプルは、以下の操作方法を示しています。

- トピック (FIFO または非 FIFO) を作成します。
- フィルターを適用するオプションを使用して、複数のキューをトピックにサブスクライブします。
- メッセージをトピックに発行します。
- キューをポーリングして受信メッセージを確認します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

これは、このシナリオのエントリポイントです。

```
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();
};
```

前述のコードは、必要な依存関係を提供し、シナリオを開始します。次のセクションには、この例の大部分が含まれています。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - 公開
  - ReceiveMessage

- SetQueueAttributes
- Subscribe
- Unsubscribe

API Gateway を使用して Lambda 関数を呼び出す

次のコード例は、Amazon API Gateway によって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

Lambda JavaScript ランタイム API を使用して AWS Lambda 関数を作成する方法を示します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、Amazon API Gateway によって呼び出される Lambda 関数を作成する方法を示します。この関数は、Amazon DynamoDB テーブルをスキャンして、Amazon Simple Notification Service (Amazon SNS) を使用して、従業員に年間の記念日を祝福するテキストメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

スケジュールされたイベントを使用した Lambda 関数の呼び出し

次のコード例は、Amazon EventBridge スケジュールされたイベントによって呼び出される AWS Lambda 関数を作成する方法を示しています。

SDK for JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールする

ように EventBridge を設定します。この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

サーバーレスサンプル

Amazon SNS トリガーから Lambda 関数を呼び出す

次のコード例は、SNS トピックからメッセージを受信することによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行する方法を確認してください。

JavaScript を使用した Lambda での SNS イベントの消費。

```
// 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;
  }
}
```

TypeScript を使用した Lambda での SNS イベントの消費。

```
// 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;
  }
```

```
}
```

SDK for JavaScript (v3) を使用した Amazon SQS の例

次のコード例は、Amazon SQS で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

Hello Amazon SQS

次のコード例は、Amazon SQS の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS クライアントを初期化し、キューを一覧表示します。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListQueues」を参照してください。

トピック

- アクション
- シナリオ
- サーバーレスサンプル

アクション

`ChangeMessageVisibility`

次の例は、ChangeMessageVisibility を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS メッセージを受信し、可視性タイムアウトを変更します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ChangeMessageVisibility」を参照してください。

`CreateQueue`

次の例は、`CreateQueue` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS 標準キューを作成します。

```
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;
};
```

ロングポーリングでキューを作成します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateQueue」を参照してください。

`DeleteMessage`

次の例は、`DeleteMessage` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS メッセージを受信および削除します。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteMessage」を参照してください。

`DeleteMessageBatch`

次の例は、`DeleteMessageBatch` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteMessageBatch」を参照してください。

DeleteQueue

次の例は、DeleteQueue を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューを削除します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteQueue」を参照してください。

`GetQueueAttributes`

次の例は、`GetQueueAttributes` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「GetQueueAttributes」を参照してください。

`GetQueueUrl`

次の例は、`GetQueueUrl` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューの URL を取得します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetQueueUrl」を参照してください。

`ListQueues`

次の例は、`ListQueues` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューを一覧表示します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListQueues」を参照してください。

`ReceiveMessage`

次の例は、`ReceiveMessage` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューからメッセージを受信します。

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

ロングポーリングサポートを使用して Amazon SQS キューからメッセージを受信します。

```
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;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ReceiveMessage」を参照してください。

SendMessage

次の例は、SendMessage を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューにメッセージを送信します。

```
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;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SendMessage」を参照してください。

SetQueueAttributes

次の例は、SetQueueAttributes を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
};
```

ロングポーリングを使用するように Amazon SQS キューを設定します。

```
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;
};
```

デッドレターキューを設定します。

```
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;
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「SetQueueAttributes」を参照してください。

シナリオ

Amazon Textract エクスプローラーアプリケーションを作成する

次のコード例は、インタラクティブアプリケーションを使用して Amazon Textract 出力を調べる方法を示しています。

SDK for JavaScript (v3)

を使用して、Amazon Textract を使用してドキュメントイメージからデータを抽出し、インタラクティブなウェブページに表示する React アプリケーション AWS SDK for JavaScript を構築する方法を示します。この例はウェブブラウザで実行され、認証情報に認証された Amazon Cognito ID が必要です。Amazon Simple Storage Service (Amazon S3) をストレージに使用し、通知のために、Amazon Simple Notification Service (Amazon SNS) トピックにサブスクライブした Amazon Simple Queue Service (Amazon SQS) キューをポーリングします。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

メッセージをキューに発行する

次のコードサンプルは、以下の操作方法を示しています。

- トピック (FIFO または非 FIFO) を作成します。
- フィルターを適用するオプションを使用して、複数のキューをトピックにサブスクライブします。
- メッセージをトピックに発行します。
- キューをポーリングして受信メッセージを確認します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

これは、このシナリオのエントリポイントです。

```
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();
};
```

前述のコードは、必要な依存関係を提供し、シナリオを開始します。次のセクションには、この例の大部分が含まれています。

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。

- CreateQueue
- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- 公開
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

サーバーレスサンプル

Amazon SQS トリガーから Lambda 関数を呼び出す

次のコード例では、SQS キューからメッセージを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での SQS イベントの消費。

```
// 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;
  }
}
```

TypeScript を使用した Lambda での SQS イベントの消費。

```
// 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;
  }
}
```

Amazon SQS トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、SQS キューからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での SQS バッチアイテム失敗のレポート。

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

TypeScript を使用して Lambda で SQS バッチ項目の失敗を報告します。

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

SDK for JavaScript (v3) を使用した Step Functions の例

次のコード例は、Step Functions で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`StartExecution`

次の例は、`StartExecution` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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" });
}
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「StartExecution」を参照してください。

AWS STS SDK for JavaScript (v3) を使用した例

次のコード例は、 で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています AWS STS。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`AssumeRole`

次の例は、`AssumeRole` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成します。

```
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 });
```

IAM ロールを割り当てます。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AssumeRole」を参照してください。

サポート SDK for JavaScript (v3) を使用した例

次のコード例は、 で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています サポート。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

こんにち サポートは

次のコード例は、 サポートの使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`main()` を呼び出してサンプルを実行します。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeServices」を参照してください。

トピック

- 基本
- アクション

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- ケースの利用可能なサービスと重要度レベルを取得して表示する方法
- 選択したサービス、カテゴリ、重要度レベルを使用してサポートケースを作成する方法
- 当日のオープンケースのリストを取得して表示する方法
- 新しいケースに添付セットとコミュニケーションを追加する方法
- ケースの新しい添付ファイルとコミュニケーションについて説明する方法
- ケースを解決する方法

- 当日の解決済みケースのリストを取得して表示する方法

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ターミナルでインタラクティブシナリオを実行します。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases

- DescribeCommunications
- DescribeServices
- DescribeSeverityLevels
- ResolveCase

アクション

`AddAttachmentsToSet`

次の例は、`AddAttachmentsToSet` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AddAttachmentsToSet」を参照してください。

AddCommunicationToCase

次の例は、AddCommunicationToCase を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「AddCommunicationToCase」を参照してください。

`CreateCase`

次の例は、`CreateCase` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateCase」を参照してください。

`DescribeAttachment`

次の例は、`DescribeAttachment` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeAttachment」を参照してください。

`DescribeCases`

次の例は、`DescribeCases` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeCases」を参照してください。

`DescribeCommunications`

次の例は、`DescribeCommunications` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeCommunications」を参照してください。

`DescribeSeverityLevels`

次の例は、`DescribeSeverityLevels` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
```

```
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeSeverityLevels」を参照してください。

`ResolveCase`

次の例は、`ResolveCase` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ResolveCase」を参照してください。

SDK for JavaScript (v3) を使用した Systems Manager の例

次のコード例は、Systems Manager で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

開始方法

こんにちは、Systems Manager

次のコード例は、Systems Manager の使用を開始する方法を示しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListDocuments」を参照してください。

トピック

- 基本
- アクション

基本

基本を学ぶ

次のコードサンプルは、以下の操作方法を示しています。

- メンテナンスウィンドウを作成します。
- メンテナンスウィンドウのスケジュールを変更します。
- ドキュメントを作成します。
- 指定された EC2 インスタンスにコマンドを送信します。
- 新しい OpsItem を作成します。
- OpsItem を更新して解決します。
- メンテナンスウィンドウ、OpsItem、およびドキュメントを削除します。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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 });
}
```

- API の詳細については、『AWS SDK for JavaScript API リファレンス』の以下のトピックを参照してください。
  - CreateDocument

- CreateMaintenanceWindow
- CreateOpsItem
- DeleteMaintenanceWindow
- ListCommandInvocations
- SendCommand
- UpdateOpsItem

アクション

`CreateDocument`

次の例は、`CreateDocument` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「CreateDocument」を参照してください。

`CreateMaintenanceWindow`

次の例は、`CreateMaintenanceWindow` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「CreateMaintenanceWindow」を参照してください。

CreateOpsItem

次の例は、CreateOpsItem を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateOpsItem」を参照してください。

`DeleteDocument`

次の例は、`DeleteDocument` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「DeleteDocument」を参照してください。

`DeleteMaintenanceWindow`

次の例は、`DeleteMaintenanceWindow` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API Reference」の「DeleteMaintenanceWindow」を参照してください。

`DescribeOpsItems`

次の例は、`DescribeOpsItems` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeOpsItems」を参照してください。

`ListCommandInvocations`

次の例は、`ListCommandInvocations` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListCommandInvocations」を参照してください。

SendCommand

次の例は、SendCommand を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「SendCommand」を参照してください。

`UpdateMaintenanceWindow`

次の例は、`UpdateMaintenanceWindow` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateMaintenanceWindow」を参照してください。

`UpdateOpsItem`

次の例は、UpdateOpsItem を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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;
    }
```

```
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateOpsItem」を参照してください。

SDK for JavaScript (v3) を使用した Amazon Textract の例

次のコード例は、Amazon Textract で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- シナリオ

シナリオ

Amazon Textract エクスプローラーアプリケーションを作成する

次のコード例は、インタラクティブアプリケーションを使用して Amazon Textract 出力を調べる方法を示しています。

SDK for JavaScript (v3)

を使用して、Amazon Textract を使用してドキュメントイメージからデータを抽出し、インタラクティブなウェブページに表示する React アプリケーション AWS SDK for JavaScript を構築する方法を示します。この例はウェブブラウザで実行され、認証情報に認証された Amazon Cognito ID が必要です。Amazon Simple Storage Service (Amazon S3) をストレージに使用し、通知のために、Amazon Simple Notification Service (Amazon SNS) トピックにサブスクライブした Amazon Simple Queue Service (Amazon SQS) キューをポーリングします。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

顧客からのフィードバックを分析するアプリケーションの作成

次のコード例は、顧客のコメントカードを分析し、元の言語から翻訳し、顧客の感情を判断し、翻訳されたテキストから音声ファイルを生成するアプリケーションの作成方法を示しています。

SDK for JavaScript (v3)

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、GitHub のプロジェクトを参照してください。次の抜粋 AWS SDK for JavaScript は、Lambda 関数内で がどのように使用されるかを示しています。

```
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 };
};
```

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

SDK for JavaScript (v3) を使用した Amazon Transcribe の例

次のコード例は、Amazon Transcribe で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

アクション

`DeleteMedicalTranscriptionJob`

次の例は、`DeleteMedicalTranscriptionJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成します。

```
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 };
```

医療分野の文字起こしジョブを削除します。

```
// 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();
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteMedicalTranscriptionJob」を参照してください。

`DeleteTranscriptionJob`

次の例は、`DeleteTranscriptionJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

文字起こしジョブを削除します。

```
// 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();
```

クライアントを作成します。

```
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 };
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteTranscriptionJob」を参照してください。

`ListMedicalTranscriptionJobs`

次の例は、`ListMedicalTranscriptionJobs` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成します。

```
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 };
```

医療分野の文字起こしジョブを一覧表示します。

```
// 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();
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListMedicalTranscriptionJobs」を参照してください。

`ListTranscriptionJobs`

次の例は、`ListTranscriptionJobs` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

文字起こしジョブを一覧表示します。

```
// 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();
```

クライアントを作成します。

```
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 };
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「ListTranscriptionJobs」を参照してください。

`StartMedicalTranscriptionJob`

次の例は、`StartMedicalTranscriptionJob` を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

クライアントを作成します。

```
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 };
```

医療分野の文字起こしジョブを開始します。

```
// 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();
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartMedicalTranscriptionJob」を参照してください。

StartTranscriptionJob

次の例は、StartTranscriptionJob を使用する方法を説明しています。

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

文字起こしジョブを開始します。

```
// 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();
```

クライアントを作成します。

```
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 };
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartTranscriptionJob」を参照してください。

シナリオ

Amazon Transcribe ストリーミングアプリケーションを構築する

次のコード例は、ライブ音声をリアルタイムで記録、転写、翻訳し、結果を E メールで送信するアプリケーションを構築する方法を示しています。

SDK for JavaScript (v3)

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、転写、翻訳し、Amazon Simple Email Service (Amazon SES) を使用して結果を E メールで送信するアプリケーションを構築する方法について説明します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

SDK for JavaScript (v3) を使用した Amazon Translate の例

次のコード例は、Amazon Translate で AWS SDK for JavaScript (v3) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

シナリオ

Amazon Transcribe ストリーミングアプリケーションを構築する

次のコード例は、ライブ音声をリアルタイムで記録、転写、翻訳し、結果を E メールで送信するアプリケーションを構築する方法を示しています。

SDK for JavaScript (v3)

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、転写、翻訳し、Amazon Simple Email Service (Amazon SES) を使用して結果を E メールで送信するアプリケーションを構築する方法について説明します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Amazon Lex chatbot を構築する

次のコード例は、ウェブサイトの訪問者を引き付けるチャットボットを作成する方法を示しています。

SDK for JavaScript (v3)

ウェブアプリケーション内に Amazon Lex chatbotを作成して、ウェブサイトの訪問者に対応することができます。

完全なソースコードとセットアップと実行の手順については、 デ AWS SDK for JavaScript ベロッパーガイドのAmazon Lexチャットボットの構築」の完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

顧客からのフィードバックを分析するアプリケーションの作成

次のコード例は、顧客のコメントカードを分析し、元の言語から翻訳し、顧客の感情を判断し、翻訳されたテキストから音声ファイルを生成するアプリケーションの作成方法を示しています。

SDK for JavaScript (v3)

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、GitHub のプロジェクトを参照してください。次の抜粋 AWS SDK for JavaScript は、Lambda 関数内で がどのように使用されるかを示しています。

```
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 };
};
```

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

この AWS 製品またはサービスのセキュリティ

クラウドセキュリティは Amazon Web Services (AWS) の最優先事項です。 AWS のお客様は、セキュリティを非常に重視する組織の要件を満たせるように構築されたデータセンターとネットワークアーキテクチャーから利点を得ます。セキュリティは、 AWS お客様とお客様の間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

クラウドのセキュリティ – AWS クラウドで提供されているすべてのサービスを実行するインフラストラクチャ AWS を保護し、安全に使用できるサービスを提供します。における当社のセキュリティ責任は最優先事項であり AWS、当社のセキュリティの有効性は、AWS コンプライアンスプログラムの一環としてサードパーティーの監査者によって定期的にテストおよび検証されます。

クラウド内のセキュリティ – お客様の責任は、使用している AWS サービス、データの機密性、組織の要件、適用される法律や規制などのその他の要因によって決まります。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて責任共有モデルに従います。 AWS サービスセキュリティ情報については、AWS 「サービスセキュリティドキュメント」ページとAWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービスを参照してください。

トピック

- この AWS 製品またはサービスのデータ保護
- Identity and Access Management
- この AWS 製品またはサービスのコンプライアンス検証
- この AWS 製品またはサービスの耐障害性
- この AWS 製品またはサービスのインフラストラクチャセキュリティ
- 最小 TLS バージョンを強制する

この AWS 製品またはサービスのデータ保護

責任 AWS 共有モデル、この AWS 製品またはサービスのデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライ

バシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された AWS 責任共有モデルおよび GDPR のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、その中のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、AWS CLIまたは SDK を使用して、この AWS 製品またはサービスまたは他の AWS のサービス を使用する場合も同様です。 AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS リソースの使用を許可する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、 で行う作業によって異なります AWS。

サービスユーザー – AWS のサービス を使用してジョブを実行する場合、管理者から必要な認証情報とアクセス許可が提供されます。さらに多くの AWS 機能を使用して作業を行う場合は、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。の機能にアクセスできない場合は AWS、 AWS のサービス AWS ID とアクセスのトラブルシューティング「」または使用している のユーザーガイドを参照してください。

サービス管理者 – 社内の AWS リソースを担当している場合は、通常、 へのフルアクセスがあります AWS。サービスユーザーがどの AWS 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で IAM を使用する方法の詳細については AWS、使用している AWS のサービス のユーザーガイドを参照してください。

IAM 管理者 - 管理者は、 AWSへのアクセスを管理するポリシーの書き込み方法の詳細について確認する場合があります。IAM で使用できる AWS アイデンティティベースのポリシーの例を表示するには、 AWS のサービス 使用している のユーザーガイドを参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることによって、認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン

認証、Google または Facebook 認証情報は、フェデレーション ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の「 へのサインイン AWS アカウント方法」を参照してください。

AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストを暗号化して署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに自分で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。たとえば、 では、アカウントのセキュリティを高めるために多要素認証 (MFA) を使用する AWS ことをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、ID プロバイダーとのフェデレーションを使用して一時的な認証情報 AWS のサービス を使用して にアクセスすることを要求します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、、AWS Directory Serviceアイデンティティセンターディレクトリ、または ID ソースを介して提供された認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッドアイデ

ンティティがアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成するか、独自の ID ソースのユーザーとグループのセットに接続して同期し、すべての AWS アカウント とアプリケーションで使用できます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内の ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内の ID です。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替えることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(プロキシとしてロールを使用する代わりに) リソースに直接ポリシーをアタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス — 一部の は他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用してアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする と組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」を参照してください。

- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的でない追加のポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) の最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の AWS アカウント をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定するために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースのアクセス許可を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、 を含む ID の有効なアクセス許可に影響を与える可能性があります。RCP をサポートする のリストを含む Organizations と RCP の詳細については、AWS Organizations RCPs「リソースコントロールポリシー (RCPs」を参照してください。 AWS のサービス
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

IAM AWS のサービス の操作方法

ほとんどの IAM 機能と AWS のサービス の連携方法の概要については、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

IAM AWS のサービス で特定の を使用する方法については、関連するサービスのユーザーガイドのセキュリティセクションを参照してください。

AWS ID とアクセスのトラブルシューティング

次の情報は、 および IAM の使用時に発生する可能性がある一般的な問題の診断 AWS と修正に役立ちます。

トピック

- でアクションを実行する権限がありません AWS
- iam:PassRole を実行する権限がありません
- 自分の 以外のユーザーに自分の AWS リソース AWS アカウント へのアクセスを許可したい

でアクションを実行する権限がありません AWS

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要なawes:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

この場合、awes:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がありません

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して AWSにロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して AWSでアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに自分の AWS リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- がこれらの機能 AWS をサポートしているかどうかを確認するには、「」を参照してくださいIAM AWS のサービス の操作方法。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。

- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

この AWS 製品またはサービスのコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、AWS のサービス「コンプライアンスプログラムによる対象範囲内」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「 Compliance ProgramsAssurance」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供します。

- セキュリティのコンプライアンスとガバナンス – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。
- HIPAA 対応サービスのリファレンス – HIPAA 対応サービスの一覧が提供されています。すべてが HIPAA AWS のサービス の対象となるわけではありません。
- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) など) のセキュリティコントロールを保護し、そのガイダンスに AWS のサービス マッピングするためのベストプラクティスをまとめています。

- 「 デベロッパーガイド」の「ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – 不審なアクティビティや悪意のあるアクティビティがないか環境をモニタリングすることで AWS アカウント、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて責任共有モデルに従います。 AWS サービスセキュリティ情報については、AWS 「サービスセキュリティドキュメント」ページとAWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービスを参照してください。

この AWS 製品またはサービスの耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。

AWS リージョン は、低レイテンシー、高スループット、および高度に冗長なネットワークで接続された複数の物理的に分離および分離されたアベイラビリティーゾーンを提供します。

アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて責任共有モデルに従います。 AWS サービスセキュリティ情報については、AWS 「サービスセキュリティドキュメント」ページとAWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービスを参照してください。

この AWS 製品またはサービスのインフラストラクチャセキュリティ

この AWS 製品またはサービスはマネージドサービスを使用するため、 グローバルネットワークセキュリティによって AWS 保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由でこの AWS 製品またはサービスにアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードはJava 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。またはAWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて責任共有モデルに従います。 AWS サービスセキュリティ情報については、AWS 「サービスセキュリティドキュメント」ページとAWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービスを参照してください。

最小 TLS バージョンを強制する

AWS サービスと通信するときにセキュリティを強化するには、TLS 1.2 以降を使用する AWS SDK for JavaScript ように を設定します。

Transport Layer Security (TLS) は、ネットワーク上で交換されるデータのプライバシーと整合性を確保するために、ウェブブラウザやその他のアプリケーションで使用されるプロトコルです。

> **Important**
>
> 2024 年 6 月 10 日現在、TLS 1.3 が各 AWS リージョン AWS のサービス API エンドポイントで利用可能であることを発表しました。v3 AWS SDK for JavaScript は TLS バージョン自体をネゴシエートしません。代わりに、`https.Agent` を介して設定可能な Node.js によって決定される TLS バージョンを使用します。 AWS では、現在の Active LTS バージョンの Node.js を使用することをお勧めします。

Node.js での TLS の検証と適用

Node.js AWS SDK for JavaScript で を使用する場合、基盤となる Node.js セキュリティレイヤーを使用して TLS バージョンを設定します。

Node.js 12.0.0 以降では、TLS 1.3 をサポートする OpenSSL 1.1.1b 以降のバージョンが使用されます。Node.js は、使用可能な場合はデフォルトで TLS 1.3 を使用します。必要に応じて、別のバージョンを明示的に指定できます。

OpenSSL および TLS のバージョンを検証します。

コンピュータ上の Node.js で使用されている OpenSSL のバージョンを取得するには、次のコマンドを実行します。

```
node -p process.versions
```

リスト内の OpenSSL のバージョンは、次の例に示すように、Node.js で使用されるバージョンです。

```
openssl: '1.1.1b'
```

コンピュータ上の Node.js で使用されている TLS のバージョンを取得するには、Node シェルを起動し、次のコマンドを順に実行します。

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

最後のコマンドは、次の例に示すように TLS のバージョンを出力します。

```
'TLSv1.3'
```

Node.js はデフォルトでこのバージョンの TLS を使用し、呼び出しが失敗した場合は別のバージョンの TLS のネゴシエートを試みます。

サポートされる TLS バージョンの最小バージョンと最大バージョンの確認

開発者は、次のスクリプトを使用して Node.js でサポートされている最小および最大 TLS バージョンを確認できます。

```
import tls from "tls";
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +
 tls.DEFAULT_MAX_VERSION);
```

最後のコマンドは、次の例に示すようにデフォルトの最小および最大の TLS のバージョンを出力します。

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

TLS の最小バージョンの指定

Node.js は、呼び出しが失敗した場合に TLS のバージョンをネゴシエートします。コマンドラインからスクリプトを実行するとき、または JavaScript コードのリクエストごとに、このネゴシエーション中に TLS の最小バージョンを指定できます。

コマンドラインから TLS の最小バージョンを指定するには、Node.js バージョン 11.4.0 以降を使用する必要があります。特定の Node.js バージョンをインストールするには、まず「Node Version Managerのインストールと更新」のステップを使用して、Node Version Manager(nvm)をインストールします。続いて、次のコマンドを実行し、特定バージョンの Node.js をインストールして使用します。

```
nvm install 11
nvm use 11
```

Enforce TLS 1.2

TLS 1.2 が最小許容バージョンであることを指定するには、次の例に示すように、スクリプトの実行時に `--tls-min-v1.2` 引数を指定します。

```
node --tls-min-v1.2 yourScript.js
```

JavaScript コード内の特定のリクエストに対して最小許容 TLS バージョンを指定するには、次の例に示すように、minVersion パラメータを使用してプロトコルを指定します。

```
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

TLS 1.3 が最小許容バージョンであることを指定するには、次の例に示すように、スクリプトの実行時に --tls-min-v1.3 引数を指定します。

```
node --tls-min-v1.3 yourScript.js
```

JavaScript コード内の特定のリクエストに対して最小許容 TLS バージョンを指定するには、次の例に示すように、minVersion パラメータを使用してプロトコルを指定します。

```
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'
            }
```

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

ブラウザスクリプトでの TLS の検証と適用

ブラウザスクリプトで SDK for JavaScript を使用する場合、ブラウザの設定によって、使用される TLS のバージョンが制御されます。ブラウザで使用される TLS のバージョンは、スクリプトによって検出または設定できないため、ユーザーが設定する必要があります。ブラウザスクリプトで使用される TLS のバージョンを検証して適用する方法については、お使いのブラウザの手順を参照してください。

Microsoft Internet Explorer

1. Internet Explorer を開きます。
2. メニューバーから、[ツール] - [インターネットオプション] - [詳細設定] タブを選択します。
3. [セキュリティ] まで下にスクロールし、[TLS 1.2 の使用] チェックボックスを手動でオンにします。
4. [OK] をクリックします。
5. ブラウザを閉じて、Internet Explorer を再起動します。

Microsoft Edge

1. Windows メニューの検索ボックスに、「############」と入力します。
2. [最も一致する検索結果] で、[インターネットオプション] をクリックします。
3. [インターネットのプロパティ] ウィンドウの [詳細設定] タブで、[セキュリティ] セクションまで下にスクロールします。
4. [TLS 1.2 の使用] チェックボックスをオンにします。
5. [OK] をクリックします。

Google Chrome

1. Google Chrome を開きます。
2. Alt + F キーを押し、[設定] を選択します。
3. 下にスクロールし、[詳細設定] を選択します。

4. [システム] まで下にスクロールし、[パソコンのプロキシ設定を開く] をクリックします。
5. [詳細設定] タブを選択します。
6. [セキュリティ] まで下にスクロールし、[TLS 1.2 の使用] チェックボックスを手動でオンにします。
7. [OK] をクリックします。
8. ブラウザを閉じて Google Chrome を再起動します。

Mozilla Firefox

1. Firefox を開きます。
2. アドレスバーに「about:config」と入力し、Enter キーを押します。
3. [検索] フィールドに「tls」と入力します。[security.tls.version.min] のエントリを見つけてダブルクリックします。
4. TLS 1.2 プロトコルをデフォルトに指定するには、整数値を 3 に設定します。
5. [OK] をクリックします。
6. ブラウザを閉じて Mozilla Firefox を再起動します。

Apple Safari

SSL プロトコルを有効にするオプションはありません。Safari バージョン 7 以降を使用している場合は、TLS 1.2 が自動的に有効になります。

v3 リクエストでの TLS AWS SDK for JavaScript バージョンの取得

AWS SDK リクエストで使用される TLS バージョンは、次のスクリプトを使用してログに記録できます。

```
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);
});
```

次の例に示すように、最後のコマンドは使用中の TLS バージョンを出力します。

```
Using TLS version: TLSv1.3
```

のバージョン 2.x から 3.x への移行 AWS SDK for JavaScript

AWS SDK for JavaScript バージョン 3 はバージョン 2 の大幅な書き換えです。このセクションでは、2 つのバージョンの違いについて説明し、SDK for JavaScript のバージョン 2 からバージョン 3 に移行する方法について説明します。

codemod を使用してコードを SDK for JavaScript v3 に移行する

AWS SDK for JavaScript バージョン 3 (v3) には、認証情報、Amazon S3 マルチパートアップロード、DynamoDB ドキュメントクライアント、ウェーターなど、クライアント設定とユーティリティ用のモダナイズされたインターフェイスが付属しています。v2 で何が変更されたか、および各変更の v3 に相当するものについては、AWS SDK for JavaScript GitHub リポジトリの移行ガイドを参照してください。

v3 AWS SDK for JavaScript を最大限に活用するには、以下で説明する codemod スクリプトを使用することをお勧めします。

codemod を使用して既存の v2 コードを移行する

aws-sdk-js-codemod の codemod スクリプトのコレクションは、既存の AWS SDK for JavaScript (v2) アプリケーションを v3 APIs を使用するように移行するのに役立ちます。次のように変換を実行できます。

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

例えば、v2 から Amazon DynamoDB クライアントを作成し、`listTables` オペレーションを呼び出す次のコードがあるとします。

```
// 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);
```

example.ts に対する v2-to-v3 変換は次のように実行できます。

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

次のように、DynamoDB インポートを v3 に変換し、v3 クライアントを作成して、listTables オペレーションを呼び出します。

```
// 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);
```

一般的なユースケースの変換を実装しました。コードが正しく変換されない場合は、入力コードの例と確認された/期待される出力コードを含むバグレポートまたは機能リクエストを作成してください。特定のユースケースが既存の問題ですでに報告されている場合は、賛成票を投じて支持を示してください。

バージョン 3 の新機能とは

SDK for JavaScript (v3) のバージョン 3 には、次の新機能が含まれています。

モジュール化されたパッケージ

ユーザーは、サービスごとに個別のパッケージを使用できます。

新しいミドルウェアスタック

ユーザーはミドルウェアスタックを使用して、オペレーション呼び出しのライフサイクルを制御できます。

さらに、SDKはTypeScriptで記述されており、静的型付けなどの多くの利点があります。

> **Important**
>
> このガイドの v3 のコード例は、ECMAScript 6 (ES6) で記述されています。ES6 は、コードをよりモダンで読みやすくし、より多くのことをおこなうための新しい構文と新機能を提供

します。ES6 では Node.js バージョン 13.x 以降を使用する必要があります。Node.js の最新バージョンをダウンロードしてインストールするには、Node.js downloads.を参照してください。詳細については、「JavaScript ES6/CommonJS 構文」を参照してください。

モジュール化されたパッケージ

SDK for JavaScript (v2) のバージョン 2 では、次のように AWS SDK 全体を使用する必要があります。

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

アプリケーションが多くの AWS サービスを使用している場合、SDK 全体をロードしても問題ありません。ただし、少数のサービスのみを使用する必要がある場合は AWS 、不要または使用しないコードでアプリケーションのサイズを増やすことを意味します。

v3 では、必要な個々の AWS サービスのみをロードして使用できます。これを次の例に示します。これにより、Amazon DynamoDB (DynamoDB) にアクセスできます。

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

個々の AWS サービスをロードして使用できるだけでなく、必要なサービスコマンドのみをロードして使用することもできます。これを次の例でDynamoDB クライアントとListTablesCommandコマンドにアクセスできることを示しています。

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

Important

サブモジュールをモジュールにインポートしないでください。例えば、次のコードはエラーになる可能性があります。

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

正しいコードは、次のとおりです。

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

コードサイズの比較

バージョン 2 (v2) では、us-west-2リージョン内のすべての Amazon DynamoDB テーブルを一覧表示するシンプルなコード例は、次のようになります。

```
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 は次のようになります。

```
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)
}
```

aws-sdkのパッケージは、アプリケーションに約40MBを追加します。var AWS = require("aws-sdk")をimport {DynamoDB} from "@aws-sdk/client-dynamodb"に置き換えることで、そのオーバーヘッドを約 3 MB に削減します。インポートを DynamoDB クライアントとListTablesCommandのコマンドだけに限定することで、オーバーヘッドを 100 KB 以下に削減します。

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

v3 でのコマンドの呼び出し

v2 または v3 コマンドを使用して v3 でオペレーションを実行できます。v3 コマンドを使用するには、コマンドと必要な AWS サービスパッケージクライアントをインポートし、非同期/待機パターンを使用して .sendメソッドを使用して コマンドを実行します。

v2 コマンドを使用するには、必要な AWS サービスパッケージをインポートし、コールバックまたは非同期/待機パターンを使用してパッケージ内で v2 コマンドを直接実行します。

v3 コマンドの使用

v3 には、各 AWS サービスパッケージの一連のコマンドが用意されており、その AWS サービスのオペレーションを実行できます。 AWS のサービスをインストールした後、node-modules/@aws-sdk/client-*PACKAGE_NAME*/commands folder.のプロジェクトで利用可能なコマンドを参照できます。

使用したいコマンドをインポートする必要があります。例えば、次のコードは DynamoDB サービスおよびCreateTableCommandのコマンドをロードします。

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

これらのコマンドを推奨の非同期/待機パターンで呼び出すには、次の構文を使用します。

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

例えば、次の例では、推奨される非同期/待機 パターンを使用して DynamoDB テーブルを作成します。

```
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);
};
```

v2 コマンドの使用

SDK for JavaScript で v2 コマンドを使用するには、次のコードに示すように、完全な AWS サービスパッケージをインポートします。

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

推奨される非同期/待機パターンで v2 コマンドを呼び出すには、次の構文を使用します。

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

次の例では、v2 `createTable` コマンドを使用して、推奨される非同期/待機パターンを使用して DynamoDB テーブルを作成します。

```
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();
```

次の例では、v2 createBucket コマンドを使用して、コールバックパターンを使用して Amazon S3 バケットを作成します。

```
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();
```

新しいミドルウェアスタック

SDK の v2 では、イベントリスナーをリクエストにアタッチすることで、ライフサイクルの複数のステージにわたってリクエストを変更できます。このアプローチでは、リクエストのライフサイクル中に問題が発生したことをデバッグすることが困難になる可能性があります。

v3 では、新しいミドルウェアスタックを使用してオペレーション呼び出しのライフサイクルを制御できます。このアプローチには、いくつかの利点があります。スタック内の各ミドルウェアステージは、リクエストオブジェクトに変更を加えた後、次のミドルウェアステージを呼び出します。また、エラーに至るまでのミドルウェアステージが呼び出されたかを正確に確認できるため、スタック内の問題のデバッグがはるかに簡単になります。

次の例では、ミドルウェアを使用して (先ほど作成して示した) Amazon DynamoDB クライアントにカスタムヘッダーを追加します。最初の引数は呼び出すスタックの次のミドルウエアステージであるnextを受け入れる関数と、呼び出され操作に関する情報を含むオブジェクトであるcontextを受け入れる関数です。この関数は、操作とリクエストに渡されるパラメータを含むオブジェクトのargsを受け入れる関数を返します。次のミドルウェアをargsで呼び出した結果を返します

```
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));
```

v2 と AWS SDK for JavaScript v3 の違い

このセクションでは、v2 から AWS SDK for JavaScript v3 への重要な変更をキャプチャします。v3 は v2 のモジュラー書き換えであるため、v2 と v3 ではいくつかの基本概念が異なります。これらの変更については、ブログ記事で確認できます。次のブログ記事では、作業を高速化します。

- のモジュラーパッケージ AWS SDK for JavaScript
- Modular でのミドルウェアスタックの紹介 AWS SDK for JavaScript

v2 から AWS SDK for JavaScript v3 へのインターフェイスの変更の概要を以下に示します。目標は、既に使い慣れている v2 APIs と同等の v3 を簡単に見つけられるようにすることです。

トピック

- クライアントコンストラクタ
- 認証情報プロバイダ
- Amazon S3 に関する考慮事項
- DynamoDB ドキュメントクライアント
- ウェーターと署名者
- 特定のサービスクライアントに関する注意事項

クライアントコンストラクタ

このリストは v2 設定パラメータによってインデックス化されます。

- computeChecksums
  - v2: サービスがペイロードボディを受け入れるときに、ペイロードボディの MD5 チェックサムを計算するかどうか (現在 S3 でのみサポートされています) 。
  - v3: S3 (PutObject、PutBucketCors など) の該当するコマンドは、リクエストペイロードの MD5 チェックサムを自動的に計算します。コマンドの ChecksumAlgorithmパラメータで別のチェックサムアルゴリズムを指定して、別のチェックサムアルゴリズムを使用することもできます。詳細については、S3 機能のお知らせを参照してください。
- convertResponseTypes
  - v2: レスポンスデータを解析するときに型を変換するかどうか。
  - v3: 非推奨。このオプションは、タイムスタンプや base64 バイナリなどのタイプを JSON レスポンスから変換しないため、型安全ではないと見なされます。
- correctClockSkew
  - v2: クライアントクロックの歪曲が原因で失敗したクロックスキュー修正リクエストと再試行リクエストを適用するかどうか。
  - v3: 非推奨。SDK は常にクロックスキュー修正を適用します。
- systemClockOffset
  - v2: すべての署名時間に適用するミリ秒単位のオフセット値。
  - v3: 変更なし。
- credentials
  - v2: AWS リクエストに署名するための認証情報。
  - v3: 変更なし。また、認証情報を返す非同期関数にすることもできます。関数が を返す場合expiration (Date)、有効期限の日時が近づくと、関数は再度呼び出されます。AwsAuthInputConfig 認証情報については、「v3 API リファレンス」を参照してください。
- endpointCacheSize
  - v2: エンドポイント検出オペレーションからのエンドポイントを保存するグローバルキャッシュのサイズ。
  - v3: 変更なし。
- endpointDiscoveryEnabled
  - v2: サービスによって指定されたエンドポイントを使用して オペレーションを動的に呼び出すかどうか。
  - v3: 変更なし。
- hostPrefixEnabled
  - v2: リクエストパラメータをホスト名のプレフィックスにマーシャリングするかどうか。
  - v3: 非推奨。SDK は、必要に応じて常にホスト名プレフィックスを挿入します。

- `httpOptions`

  低レベルの HTTP リクエストに渡す一連のオプション。これらのオプションは v3 では異なる方法で集計されます。新しい を指定することで設定できますrequestHandler。Node.js ランタイムで http オプションを設定する例を次に示します。詳細については、NodeHttpHandler の v3 API リファレンスを参照してください。

  すべての v3 リクエストはデフォルトで HTTPS を使用します。カスタム httpsAgent を提供するだけで済みます。

```
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*/
    }),
});
```

  http を使用するカスタムエンドポイントを渡す場合は、httpAgent を指定する必要があります。

```
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",
});
```

  クライアントがブラウザで実行されている場合、別のオプションセットを使用できます。詳細については、FetchHttpHandler の v3 API リファレンスを参照してください。

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
```

```
const dynamodbClient = new DynamoDBClient({
    requestHandler: new FetchHttpHandler({
        requestTimeout: /* number in milliseconds */
    }),
});
```

の各オプションhttpOptionsを以下に示します。

- proxy
  - v2: リクエストをプロキシするための URL。
  - v3: Node.js のプロキシの設定に従って、 エージェントでプロキシを設定できます。
- agent
  - v2: HTTP リクエストを実行する エージェントオブジェクト。接続プーリングに使用されます。
  - v3: 上記の例httpsAgentに示すように、 httpAgentまたは を設定できます。
- connectTimeout
  - v2: connectTimeoutミリ秒後にサーバーとの接続を確立できなかった後、ソケットをタイムアウトに設定します。
  - v3: connectionTimeoutは NodeHttpHandler オプションで使用できます。
- timeout
  - v2: リクエストが自動的に終了するまでにかかるミリ秒数。
  - v3: socketTimeoutは NodeHttpHandler オプションで使用できます。
- xhrAsync
  - v2: SDK が非同期 HTTP リクエストを送信するかどうか。
  - v3: 非推奨。リクエストは常に非同期です。
- xhrWithCredentials
  - v2: XMLHttpRequest オブジェクトのwithCredentials」プロパティを設定します。
  - v3: 利用できません。SDK はデフォルトのフェッチ設定を継承します。
- logger
  - v2: リクエストに関する情報をログに記録するために .write() (ストリームなど) または .log() (コンソールオブジェクトなど) に応答するオブジェクト。
  - v3: 変更なし。より詳細なログは v3 で利用できます。
- maxRedirects
  - v2: サービスリクエストに従うリダイレクトの最大量。
  - v3: 非推奨。SDK は、意図しないクロスリージョンリクエストを回避するためにリダイレクトに従いません。

- maxRetries
  - v2: サービスリクエストに対して実行する最大再試行回数。
  - v3: を に変更しましたmaxAttempts。詳細については、RetryInputConfig の v3 API リファレンス」を参照してください。は maxAttemptsである必要がありますmaxRetries + 1。
- paramValidation
  - v2: リクエストを送信する前に、入力パラメータをオペレーションの説明に対して検証するかどうか。
  - v3: 非推奨。SDK は、実行時にクライアント側で検証を行いません。
- region
  - v2: サービスリクエストを送信するリージョン。
  - v3: 変更なし。また、リージョン文字列を返す非同期関数にすることもできます。
- retryDelayOptions
  - v2: 再試行可能なエラーの再試行遅延を設定する一連のオプション。
  - v3: 非推奨。SDK は、retryStrategyクライアントコンストラクタオプションを使用して、より柔軟な再試行戦略をサポートします。詳細については、「v3 API リファレンス」を参照してください。
- s3BucketEndpoint
  - v2: 指定されたエンドポイントが個々のバケットに対応するかどうか (ルート API エンドポイントに対応する場合は失敗) 。
  - v3: を に変更しましたbucketEndpoint。詳細については、bucketEndpointの v3 API リファレンス」を参照してください。に設定するとtrue、リクエストパラメータでBucketリクエストエンドポイントを指定すると、元のエンドポイントが上書きされることに注意してください。v2 では、クライアントコンストラクタのリクエストエンドポイントがBucketリクエストパラメータを上書きします。
- s3DisableBodySigning
  - v2: 署名バージョン v4 を使用するときに S3 本文署名を無効にするかどうか。
  - v3: の名前を に変更しましたapplyChecksum。
- s3ForcePathStyle
  - v2: S3 オブジェクトのパススタイルの URLs を強制するかどうか。
  - v3: の名前を に変更しましたforcePathStyle。
- s3UseArnRegion
  - v2: リクエストされたリソースの ARN から推測されたリージョンでリクエストリージョンを上書きするかどうか。
  - v3: の名前を に変更しましたuseArnRegion。
- s3UsEast1RegionalEndpoint

  - v2: region が「us-east-1」に設定されている場合、s3 リクエストをグローバルエンドポイントに送信するか「us-east-1」リージョンエンドポイントに送信するか。
  - v3: 非推奨。リージョンが に設定されている場合、S3 クライアントは常にリージョンエンドポイントを使用しますus-east-1。リージョンを に設定aws-globalして、S3 グローバルエンドポイントにリクエストを送信できます。
- signatureCache
  - v2: でリクエストに署名する署名 (API 設定の上書き) がキャッシュされるかどうか。
  - v3: 非推奨。SDK は常にハッシュされた署名キーをキャッシュします。
- signatureVersion
  - v2: リクエストに署名する署名バージョン (API 設定の上書き) 。
  - v3: 非推奨。v2 SDK でサポートされている署名 V2 は によって廃止されました AWS。v3 は署名 v4 のみをサポートしています。
- sslEnabled
  - v2: SSL がリクエストに対して有効かどうか。
  - v3: の名前を に変更しましたtls。
- stsRegionalEndpoints
  - v2: グローバルエンドポイントまたはリージョンエンドポイントに sts リクエストを送信するかどうか。
  - v3: 非推奨。STS クライアントは、特定のリージョンに設定した場合、常にリージョンエンドポイントを使用します。STS グローバルエンドポイントaws-globalにリクエストを送信するようにリージョンを に設定できます。
- useAccelerateEndpoint
  - v2: Accelerate エンドポイントを S3 サービスで使用するかどうか。
  - v3: 変更なし。

認証情報プロバイダ

v2 では、SDK for JavaScript は、選択する認証情報プロバイダーのリストと、Node.js でデフォルトで利用可能な認証情報プロバイダーチェーンを提供します。これにより、最も一般的なプロバイダーから AWS 認証情報をロードしようとします。SDK for JavaScript v3 は、認証情報プロバイダーのインターフェイスを簡素化し、カスタム認証情報プロバイダーの使用と書き込みを容易にします。新しい認証情報プロバイダーチェーンに加えて、SDK for JavaScript v3 はすべて、v2 と同等のものを提供することを目的とした認証情報プロバイダーのリストを提供します。

v2 のすべての認証情報プロバイダーと v3 の同等の認証情報プロバイダーは次のとおりです。

デフォルトの認証情報プロバイダー

デフォルトの認証情報プロバイダーは、明示的に指定しない場合に SDK for JavaScript が AWS 認証情報を解決する方法です。

- v2: Node.js の CredentialProviderChain は、ソースからの認証情報を次の順序で解決します。
  - 環境変数
  - 共有認証情報ファイル
  - ECS コンテナの認証情報
  - 外部プロセスのスポーン
  - 指定されたファイルからの OIDC トークン
  - EC2 インスタンスメタデータ

  上記の認証情報プロバイダーの 1 つが AWS 認証情報の解決に失敗した場合、有効な認証情報が解決されるまでチェーンは次のプロバイダーにフォールバックし、すべてのプロバイダーが失敗するとチェーンはエラーをスローします。

  Browser および React Native ランタイムでは、認証情報チェーンは空であり、認証情報は明示的に設定する必要があります。
- v3: defaultProvider。認証情報のソースとフォールバック順序は v3 では変更されません。また、AWS IAM Identity Center 認証情報もサポートしています。

一時認証情報

- v2: から取得した一時的な認証情報ChainableTemporaryCredentialsを表しますAWS.STS。追加のパラメータがない場合、認証情報は AWS.STS.getSessionToken()オペレーションから取得されます。IAM ロールが指定されている場合、 AWS.STS.assumeRole()オペレーションは代わりにロールの認証情報を取得するために使用されます。 は masterCredentials と更新の処理AWS.TemporaryCredentials方法AWS.ChainableTemporaryCredentialsとは異なります。 は、STS 認証情報の連鎖をサポートするためにユーザーが渡した masterCredentials を使用して期限切れの認証情報AWS.ChainableTemporaryCredentialsを更新します。ただし、 はインスタンス化中に masterCredentials AWS.TemporaryCredentialsを再帰的に折りたたみ、中間の一時的な認証情報を必要とする認証情報を更新する機能は除外されます。

  v2 ChainableTemporaryCredentialsでは、元の TemporaryCredentialsは廃止され、 が優先されます。

- v3: fromTemporaryCredentials。@aws-sdk/credential-providers パッケージfromTemporaryCredentials()から を呼び出すことができます。例を示します。

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

Amazon Cognito ID 認証情報

Amazon Cognito Identity サービスから認証情報をロードします。通常、ブラウザで使用されます。

- v2: Amazon Cognito Identity CognitoIdentityCredentials サービスを使用して STS Web Identity フェデレーションから取得した認証情報を表します。
- v3: Cognito Identity Credential Provider @aws/credential-providersパッケージには 2 つの認証情報プロバイダー関数が用意されています。1 つは ID ID fromCognitoIdentity を受け取り を呼び出しcognitoIdentity:GetCredentialsForIdentity、もう 1 つは ID プール ID fromCognitoIdentityPoolを受け取り、最初の呼び出しcognitoIdentity:GetIdで を呼び出し、次に を呼び出しますfromCognitoIdentity。後者の呼び出しは GetId を再呼び出ししません。

  プロバイダーは、Amazon Cognito デベロッパーガイド」で説明されている「簡易フロー」を実装します。を呼び出しcognito:GetOpenIdTokenてから を呼び出す「Classic Flowsts:AssumeRoleWithWebIdentity」はサポートされていません。必要に応じて、機能リクエストを開いてください。

```
// 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 メタデータ (IMDS) 認証情報

Amazon EC2 インスタンスのメタデータサービスから受信した認証情報を表します。

- v2: EC2MetadataCredentials
- v3: fromInstanceMetadata: Amazon EC2 インスタンスメタデータサービスから認証情報を取得する認証情報プロバイダーを作成します。

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

ECS 認証情報

指定された URL から受信した認証情報を表します。このプロバイダーは、 AWS_CONTAINER_CREDENTIALS_RELATIVE_URIまたは AWS_CONTAINER_CREDENTIALS_FULL_URI環境変数で指定された URI に一時的な認証情報をリクエストします。

- v2: ECSCredentialsまたは RemoteCredentials。
- v3: Amazon ECS Container Metadata Service から認証情報を取得する認証情報プロバイダーfromContainerMetadataを作成します。

```
import { fromContainerMetadata } from "@aws-sdk/credential-providers"; // ES6 import

const client = new FooClient({
  credentials: fromContainerMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

ファイルシステムの認証情報

- v2: ディスク上の JSON ファイルからの認証情報FileSystemCredentialsを表します。
- v3: 非推奨。JSON ファイルを明示的に読み取り、クライアントに を指定できます。必要に応じて、機能リクエストを開いてください。

SAML 認証情報プロバイダー

- v2: STS SAML サポートから取得した認証情報SAMLCredentialsを表します。
- v3: 利用できません。必要に応じて、機能リクエストを開いてください。

共有認証情報ファイルの認証情報

共有認証情報ファイルから認証情報をロードします (デフォルトでは ~/.aws/credentialsまたはAWS_SHARED_CREDENTIALS_FILE環境変数で定義) 。このファイルは、さまざまな AWS SDKsとツールでサポートされています。詳細については、共有 config および認証情報ファイルのドキュメントを参照してください。

- 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
  }),
});
```

ウェブ ID 認証情報

ディスク上のファイルから OIDC トークンを使用して認証情報を取得します。これは EKS で一般的に使用されます。

- 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 },
  }),
});
```

ウェブ ID フェデレーション認証情報

STS ウェブ ID フェデレーションサポートから認証情報を取得します。

- 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 },
  }),
});
```

Amazon S3 に関する考慮事項

Amazon S3 マルチパートアップロード

v2 では、Amazon S3 クライアントには、Amazon S3 が提供するマルチパートアップロード機能を備えた大きなオブジェクトのアップロードをサポートする upload()オペレーションが含まれています。

v3 では、 @aws-sdk/lib-storageパッケージを使用できます。v2 upload()オペレーションで提供されるすべての機能をサポートし、Node.js とブラウザの両方のランタイムをサポートします。

Amazon S3 の署名付き URL

v2 では、Amazon S3 クライアントには、ユーザーが Amazon S3 からオブジェクトをアップロードまたはダウンロードするために使用できる URL を生成する getSignedUrl()および getSignedUrlPromise()オペレーションが含まれています。 Amazon S3

v3 では、 @aws-sdk/s3-request-presignerパッケージを使用できます。このパッケージには、 getSignedUrl() および getSignedUrlPromise()オペレーションの両方の 関数が含まれています。このブログ記事では、このパッケージの詳細について説明します。

Amazon S3 リージョンリダイレクト

誤ったリージョンが Amazon S3 クライアントに渡され、後続の PermanentRedirect (ステータス 301) エラーがスローされた場合、v3 の Amazon S3 クライアントはリージョンリダイレクト (以前は v2 の Amazon S3 グローバルクライアントと呼ばれていました) をサポートします。クライアント設定で followRegionRedirectsフラグを使用して、Amazon S3 クライアントにリージョンリダイレクトをフォローさせ、グローバルクライアントとしてその機能をサポートさせることができます。

> **Note**
>
> この機能は、ステータスが 301 のPermanentRedirectエラーを受信すると、失敗したリクエストが修正されたリージョンで再試行されるため、レイテンシーが増加する可能性がある

ことに注意してください。この機能は、バケットのリージョンが事前にわからない場合にのみ使用してください (複数可) 。

Amazon S3 ストリーミングとバッファされたレスポンス

v3 SDK は、潜在的に大きなレスポンスをバッファしないことを優先します。これは、v2 で を返したが、v3 で を返す Amazon S3 `GetObject`オペレーション`Stream`でよく発生します。 `Buffer`

Node.js の場合、ソケットを解放して新しいトラフィックへの接続を開いたままにするには、ストリームまたはガベージコレクションクライアントまたはそのリクエストハンドラーを使用する必要があります。

```
// 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.
```

詳細については、ソケットの枯渇に関するセクションを参照してください。

DynamoDB ドキュメントクライアント

v3 での DynamoDB ドキュメントクライアントの基本的な使用法

- v2 では、 `AWS.DynamoDB.DocumentClient` クラスを使用して、配列、数値、オブジェクトなどのネイティブ JavaScript タイプで DynamoDB APIs を呼び出すことができます。これにより、属性値の概念を抽象化することで、Amazon DynamoDB の項目の操作が簡素化されます。
- v3 では、同等の`@aws-sdk/lib-dynamodb`クライアントを使用できます。これは v3 SDK の通常のサービスクライアントに似ていますが、コンストラクタに基本的な DynamoDB クライアントが必要になる点が異なります。

例:

```
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 マーシャリング時の の値

- v2 では、オブジェクトのundefined値は DynamoDB へのマーシャリングプロセス中に自動的に省略されました。
- v3 では、 のデフォルトのマーシャリング動作が変更され@aws-sdk/lib-dynamodbました。undefined値を持つオブジェクトは省略されなくなりました。v2 の機能に合わせて、開発者は DynamoDB ドキュメントクライアントの removeUndefinedValuestrueで marshallOptionsを明示的に に設定する必要があります。

例:

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

パッケージ README では、その他の例と設定を利用できます。

ウェーターと署名者

このページでは、v3 AWS SDK for JavaScript でのウェーターと署名者の使用状況について説明します。

ウェイター

v2 では、すべてのウェーターがサービスクライアントクラスにバインドされるため、クライアントが待機する状態を設計したウェーターの入力で を指定する必要があります。たとえば、新しく作成されたバケットの準備が整うまで `waitFor("bucketExists")`を呼び出して待機する必要があります。

v3 では、アプリケーションにウェーターが必要ない場合はウェーターをインポートする必要はありません。さらに、必要な特定の状態を待つ必要があるウェーターのみをインポートできます。したがって、バンドルのサイズを減らし、パフォーマンスを向上させることができます。バケットの作成後に準備が整うのを待つ例を次に示します。

```
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 });
```

ウェイターの設定方法のすべてを、AWS SDK for JavaScript v3 のウェイターのブログ記事で確認できます。

Amazon CloudFront Signer

v2 では、 を使用して制限された Amazon CloudFront ディストリビューションにアクセスするリクエストに署名できますAWS.CloudFront.Signer。

v3 では、 @aws-sdk/cloudfront-signerパッケージに同じユーティリティが用意されています。

Amazon RDS Signer

v2 では、 を使用して Amazon RDS データベースに認証トークンを生成できますAWS.RDS.Signer。

v3 では、同様のユーティリティクラスが @aws-sdk/rds-signerパッケージで利用できます。

Amazon Polly 署名者

v2 では、 を使用して Amazon Polly サービスによって合成された音声への署名付き URL を生成できます AWS.Polly.Presigner。

v3 では、同様のユーティリティ関数が @aws-sdk/polly-request-presigner パッケージで利用できます。

特定のサービスクライアントに関する注意事項

AWS Lambda

Lambda 呼び出しレスポンスタイプは v2 と v3 で異なります。

```
// 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 チェックサム

メッセージ本文の MD5 チェックサムの計算をスキップするには、設定オブジェクトで を false md5に設定します。それ以外の場合、SDK はデフォルトでメッセージを送信するためのチェックサムを計算し、取得されたメッセージのチェックサムを検証します。

```
// 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
});
```

これを入力パラメータとする Amazon SQS オペレーションQueueUrlでカスタムを使用する場合、v2 では、Amazon SQS クライアントのデフォルトエンドポイントを上書きQueueUrlするカスタムを指定できます。

マルチリージョンメッセージ

v3 では、リージョンごとに 1 つのクライアントを使用する必要があります。 AWS リージョンは、クライアントレベルで初期化され、リクエスト間で変更されません。

```
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);
}
```

カスタムエンドポイント

v3 では、カスタムエンドポイント、つまりデフォルトのパブリック Amazon SQS エンドポイントとは異なるエンドポイントを使用する場合は、常に Amazon SQS クライアントと　QueueUrlフィールドにエンドポイントを設定する必要があります。

```
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",
});
```

カスタムエンドポイントを使用していない場合は、クライアントendpointで を設定する必要はありません。

```
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",
});
```

補足ドキュメント

次の表に、 AWS SDK for JavaScript (v3) の使用と理解に役立つ補足ドキュメントへのリンクを示します。

名前	メモ
SDK クライアント	SDK クライアントと設定可能な一般的なコンストラクタパラメータの初期化に関する情報。

名前	メモ
ノートのアップグレード (2.x から 3.x)	AWS SDK for JavaScript (v2) からのアップグレードに関する情報。
AWS Lambda Node.js ランタイムでの AWS SDK for JavaScript (v3) の使用	AWS SDK for JavaScript (v3) AWS Lambda を使用して 内で作業するためのベストプラクティス。
パフォーマンス	AWS SDK チームが SDK のパフォーマンスを最適化する方法に関する情報と、効率的に実行するように SDK を設定するためのヒントが含まれています。
TypeScript	TypeScript のヒントと AWS SDK for JavaScript (v3) に関連するFAQs。
エラー処理	AWS SDK for JavaScript (v3) に関連するエラーに対処するためのヒント。

AWS SDK for JavaScript バージョン 3 のドキュメント履歴

ドキュメント履歴

次の表は、2020年10月20日以降のAWS SDK for JavaScriptのV3リリースの重要な変更点を示しています。このドキュメントの更新に関する通知については、RSS フィードを購読してください。

変更	説明	日付
TLS 1.3 がすべてのリージョンのすべての AWS サービス API エンドポイントでサポートされるようになりました	サポートされている TLS バージョンと TLS バージョンのログ記録方法を更新しました。	2025 年 4 月 10 日
@smithy/types を使用したクライアントの生成	@smithy/types パッケージを使用したクライアントの生成で更新されたコンテンツ。	2025 年 2 月 15 日
チェックサムによるデータ整合性保護	自動チェックサム計算の詳細でコンテンツが更新されました。	2025年 1 月 15 日
Amazon S3 チェックサム	Amazon S3 でフレキシブルチェックサムを使用する方法に関するセクションを追加しました。	2025 年 1 月 1 日
DynamoDB でのアカウントベースのエンドポイントのサポート	DynamoDB のアカウントベースのエンドポイントのサポート AWS SDK for JavaScript が追加されました。	2024 年 9 月 26 日
SDK ログ記録に関する新しいトピック	SDK for JavaScript で行われた API コールをログに記録する方法を説明するトピックが追加されました。ミドルウェアを使用してリクエストをログ	2024 年 9 月 26 日

	に記録する方法に関する情報も含まれています。	
発表	Internet Explorer 11 のサポート終了をリマインドするトップバナーを更新しました。	2022 年 9 月 23 日
マイナーな更新	明確にするためにマイナーな更新を加え、壊れたリンクを解決しました。 AWS SDKsおよびツールリファレンスガイドへの意識向上リンクを追加しました。	2022 年 8 月 22 日
最小 TLS バージョンを適用する	TLS 1.3 に関する情報を追加しました。	2022 年 3 月 31 日
Node.js トピックで認証情報を設定するが更新されました	Node.js for AWS SDK for JavaScript V3 での認証情報の設定に関するトピックを更新します。	2020 年 10 月 20 日
v3 への移行	v3 AWS SDK for JavaScript への移行方法を説明するトピックを追加しました。	2020 年 10 月 20 日
使用開始	ブラウザの使用開始と Node.js for AWS SDK for JavaScript V3 の使用開始に関するトピックを更新しました。	2020 年 10 月 20 日
Browser builder (ブラウザビルダー)	AWS Browser Builder に関する情報は、 AWS SDK for JavaScript V3 に必須ではないため削除されました。	2020 年 10 月 20 日
Amazon Transcribeサービスの例が更新されました	AWS SDK for JavaScript V3 の Amazon Transcribe サービス例を更新しました。	2020 年 10 月 20 日

Amazon Simple通知サービスのサービス例が更新されました	AWS SDK for JavaScript V3 の Amazon Simple Notification Service サービス例を更新しました。	2020 年 10 月 20 日
Amazon Simple Email サービスのサービス例が更新されました	AWS SDK for JavaScript V3 の Amazon Simple Email Service サービス例を更新しました。	2020 年 10 月 20 日
Amazon Redshift サービス例が更新されました	AWS SDK for JavaScript V3 の Amazon Redshift サービス例を更新しました。	2020 年 10 月 20 日
Amazon Lex サービス例が更新されました	AWS SDK for JavaScript V3 の Amazon Lex サービス例を更新しました。	2020 年 10 月 20 日
AWS Elemental MediaConvert サービス例が更新されました	AWS SDK for JavaScript V3 AWS Elemental MediaConvert のサービス例を更新しました。	2020 年 10 月 20 日
AWS Lambda サービス例が更新されました	AWS SDK for JavaScript V3 AWS Lambda のサービス例を更新しました。	2020 年 10 月 20 日
AWS SDK for JavaScript V3 デベロッパーガイドのプレビュー	AWS SDK for JavaScript V3 デベロッパーガイドのプレリリースバージョンをリリースしました。	2020 年 10 月 19 日