SDK バージョン 3 のデベロッパーガイド

AWS SDK for JavaScript

AWS SDK for JavaScript: SDK バージョン 3 のデベロッパーガイド

Table of Contents

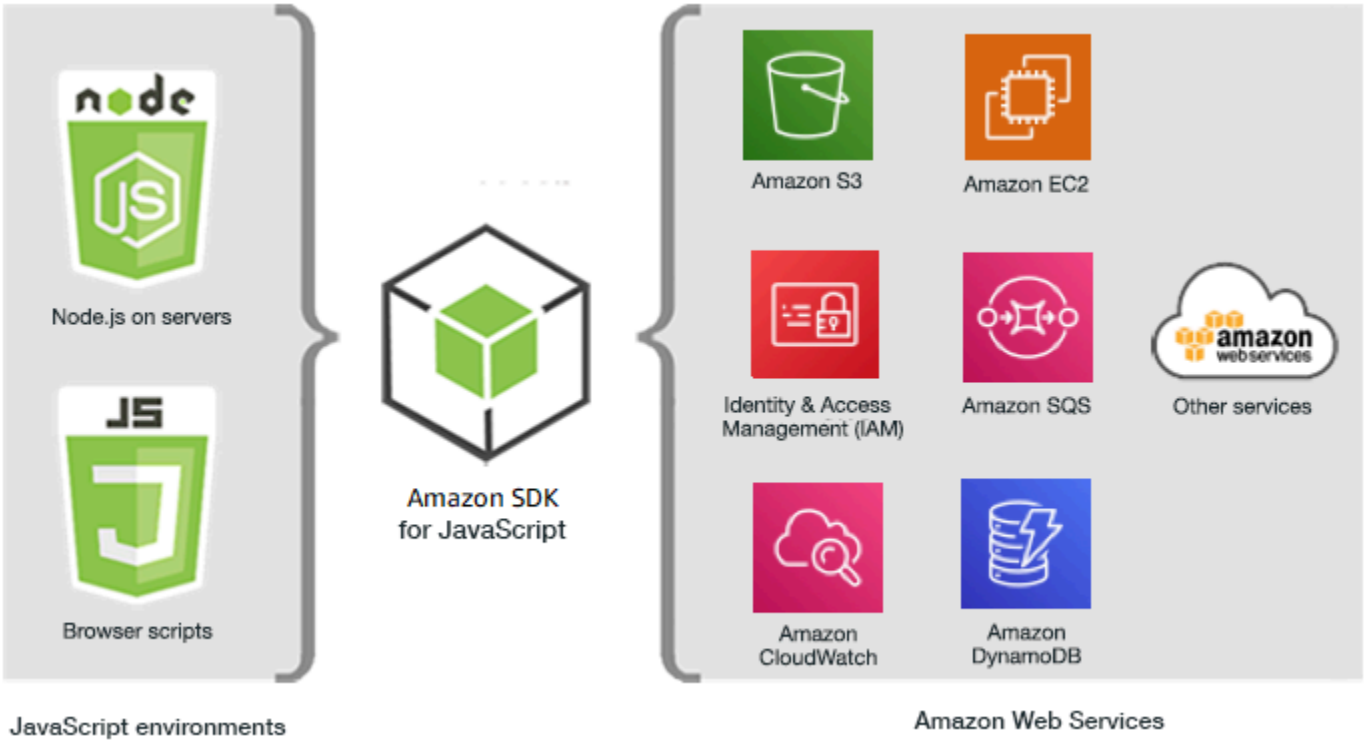
AWS SDK for JavaScript V3 APIリファレンスガイドでは、バージョン 3 (V3) のすべてのAPIオペレーションについて詳しく説明しています AWS SDK for JavaScript 。

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 リファレンスガイドは、 のサービス用の JavaScript AWS 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、「」の例に従ってください使用を開始する。

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

現在 用の SDK のバージョン 2.x を使用している場合は JavaScript、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 サービス、「」を参照してくださいSDK JavaScript (v3) コード例の。

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。

SDK for Node.js の使用は、ウェブブラウザ JavaScript で SDK for 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。

での SDK の使用 AWS Cloud9

AWS Cloud9 IDE の の SDK を使用して Node.js JavaScript アプリケーションを開発することもできます。SDK for AWS Cloud9 での の使用の詳細については、 JavaScript「」を参照してくださいAWS Cloud9 で を使用する AWS SDK for JavaScript。

での SDK の使用 AWS Amplify

ブラウザベースのウェブ、モバイルアプリケーション、ハイブリッドアプリケーションの場合は、AWS Amplify で ライブラリ GitHubを使用することもできます。SDK for を拡張し JavaScript、宣言型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、 の SDK と同じブラウザサポートを提供しない場合があります JavaScript。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。

すべての主要なウェブブラウザは、ウェブブラウザで実行されている JavaScript. JavaScript code の実行をサポートしています。多くの場合、クライアント側 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 JavaScript で SDK を に使用する方法は、Node.js 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。

V3 でブラウザーを使用

V3 を使用すると、必要な JavaScript ファイルの SDK のみをバンドルしてブラウザに含めることができるため、オーバーヘッドが軽減されます。

HTML ページで の SDK の V3 JavaScript を使用するには、Webpack を使用して必要なクライアントモジュールとすべての必要な JavaScript 関数を 1 つの JavaScript ファイルにバンドルし、HTML <head> ページの のスクリプトタグに追加する必要があります。例:

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

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

SDK for の V2 を使用するには JavaScript、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 JavaScript で SDK を に使用すると、多くの魅力的なユースケースを実現できます。SDK for 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で、ブラウザアプリケーションで構築できるモノ JavaScript に関するいくつかのアイデアを次に示します。

- 組織またはプロジェクトのニーズに合わせて、リージョンや AWS サービス間で機能にアクセスして組み合わせる サービスにカスタムコンソールを構築します。
- 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。

の例について

コード JavaScript 例リポジトリ の例については、 SDK を参照してください。 AWS

リソース

このガイドに加えて、 SDK for JavaScript Developer では以下のオンラインリソースを利用できます。

- AWS SDK for JavaScript V3 API リファレンスガイド
- AWS SDKsとツールのリファレンスガイド: AWS SDKs。
- JavaScript デベロッパーブログ
- AWS JavaScript フォーラム
- JavaScript AWS Code Catalog 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、これらの JavaScript 各 JavaScript 環境で の SDK を使用する方法を示す開始方法の演習を行います。

> **Note**
>
> AWS Cloud9 IDE の JavaScript 用 SDK を使用して、Node.js JavaScript アプリケーションとブラウザベースのアプリケーションを開発できます。Node.js 開発 AWS Cloud9 に を使用する方法の例については、「」を参照してくださいAWS Cloud9 で を使用する AWS SDK for JavaScript。

トピック

を使用した SDK 認証 AWS

を使用して開発 AWS する場合、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 サービス。 AWS リソースへのプログラムによるアクセスは、環境や利用可能な 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。

をセットアップするために、ローカルで開発していて、雇用主から認証方法が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。IAM Identity Center アクセス許可セットに接続された IAM ロールである sso_role_name値は、アプリケーションで AWS 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パッケージ (SSOや などSSOOIDC) を追加する必要はありません。

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

にアクセスするアプリケーションを実行する前に AWS 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すで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`、認証情報を入力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。

にアクセスするときは、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、人間のユーザーが AWS アカウントへのフェデレーションアクセスを提供する ID プロバイダーを使用できます。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。
- Node.js 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 "readline/promises";
```

```
import {
  S3Client,
  PutObjectCommand,
  CreateBucketCommand,
  DeleteObjectCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  GetObjectCommand,
} from "@aws-sdk/client-s3";

export async function main() {
  // A region and credentials can be declared explicitly. For example
  // `new S3Client({ region: 'us-east-1', credentials: {...} })` would
  //initialize the client with those settings. However, the SDK will
  // use your local configuration and credentials if those properties
  // are not defined here.
  const s3Client = new S3Client({});

  // Create an Amazon S3 bucket. The epoch timestamp is appended
  // to the name to make it unique.
  const bucketName = `test-bucket-${Date.now()}`;
  await s3Client.send(
    new CreateBucketCommand({
      Bucket: bucketName,
    })
  );

  // Put an object into an Amazon S3 bucket.
  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "my-first-object.txt",
      Body: "Hello JavaScript SDK!",
    })
  );

  // Read the object.
  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucketName,
      Key: "my-first-object.txt",
    })
  );
```

```
  console.log(await Body.transformToString());

  // Confirm resource deletion.
  const prompt = createInterface({
    input: process.stdin,
    output: process.stdout,
  });

  const result = await prompt.question("Empty and delete bucket? (y/n) ");
  prompt.close();

  if (result === "y") {
    // Create an async iterator over lists of objects in a bucket.
    const paginator = paginateListObjectsV2(
      { client: s3Client },
      { Bucket: bucketName }
    );
    for await (const page of paginator) {
      const objects = page.Contents;
      if (objects) {
        // For every object in each page, delete it.
        for (const object of objects) {
          await s3Client.send(
            new DeleteObjectCommand({ Bucket: bucketName, Key: object.Key })
          );
        }
      }
    }

    // Once all the objects are gone, the bucket can be deleted.
    await s3Client.send(new DeleteBucketCommand({ Bucket: bucketName }));
  }
}

// Call a function if this file was run directly. This allows the file
// to be runnable without running on import.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

サンプルコードは、 にあります GitHub。

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 JavaScript で SDK for のバージョン 3 (V3) を実行する方法を示す例について説明します。

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

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例は、 JavaScript ウェブアプリケーションで SDK for をロードして使用するための基本的なパターンを示しています。

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. 「アクセス許可の設定」で「新しい IAM ロールの作成」を選択し、「IAM ロール名StartedRole」に名前 (の取得など) を入力します。
6. 「プロパティの設定StartedPool」で、アイデンティティプール名 に名前を入力します (例: を取得) 。

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

ウェブアプリで Amazon S3 バケットへのアクセスを有効にするには、Amazon Cognito ID プール用に作成された認証されていない IAM ロール (の取得StartedRoleなど) を使用します (の取得StartedPoolなど)。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] を選択します。
3. 変更するロールの名前 (の取得など) StartedRole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の追加ページで、AmazonS3ReadOnlyAccess のチェックボックスを見つけて選択します。

> **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 [Open (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. [変更を保存] を選択します。

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、 にあります GitHub。

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、ロール名を削除します (例えば、 を取得しますStartedRole)。
- Amazon Cognito コンソール で、ID プール名を削除します (例えば、 を取得します StartedPool)。

の SDK をセットアップする JavaScript

このセクションのトピックでは、SDK でサポートされているウェブサービスにアクセスできる JavaScript ように SDK for をインストールしてロードする方法について説明します。

> **Note**
>
> React Native デベロッパーは AWS Amplify 、 を使用して で新しいプロジェクトを作成する必要があります AWS。詳細については、aws-sdk-react-nativeアーカイブを参照してください。

トピック

前提条件

Node.js をサーバーにインストールします (まだインストールしていない場合)。

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 のセットアップ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、Node.js AWS Lambda を Lambda 関数として実行します。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 AWS 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、最新のウェブブラウザをすべて AWS SDK for JavaScript サポートしています。

バージョン 3.183.0 以降では、 用 SDK 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 JavaScript 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
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 以前では、 用 SDK は次の最小バージョンをサポートする ES5 アーティファクト JavaScript を使用します。

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、 の SDK 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 JavaScript。詳細については、[AWS Amplify Documentation]を参照してください。

SDK for をインストールする JavaScript

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。

npm、Node.js パッケージマネージャー 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。

の SDK をロードする JavaScript

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。例えば、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 用に を設定する JavaScript

SDK の を使用して を使用してウェブサービスを JavaScript 呼び出す前にAPI、 を設定する必要がありますSDK。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、 の多くで共通する設定、機能、その他の基本的な概念も含まれています AWS SDKs。

このセクションのトピックでは、Node.js SDK JavaScript 用の を設定し、ウェブブラウザで JavaScript 実行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

サービスオブジェクトに設定情報を渡すSDKことで、 を設定できます。

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ー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 のコピーに自動的にマージされますAWS.config。

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、 によって管理されるグローバル設定はなくなりましたSDK。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 の で使用する各サービスは、そのサービスの の一部であるサービスオブジェクトを介して JavaScript アクセスAPIされます。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 リージョンの 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、同じ地理的エリア AWS にある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。の でサービスクライアントを作成するときにリージョンを指定SDK JavaScript し、 がそのリージョンのサービスSDKにアクセスするようにします。 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 の JavaScript は、デフォルトで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の はそれを JavaScript 読み取り、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 で使用する認証情報を保存できるのと同様にSDK、 AWS リージョンやその他の設定は、 configがSDK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はロード時にconfigファイル JavaScript 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 ファイルconfigと credentials ファイルの使用の詳細については、「」およびAWS SDKs「ツールリファレンスガイド」の「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。

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、 サービスを呼び出しているユーザーと、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。

ウェブブラウザまたは Node.js サーバーで実行されているかどうかにかかわらず、 JavaScript コードは 経由で 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API。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。

Node.js と JavaScriptウェブブラウザでは、認証情報を設定する方法がいくつかあります。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「最小特権の付与」セクション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

をセットアップするために、ローカルで開発していて、雇用主から認証方法が与えられていない新規ユーザーをお勧めします 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は、 にあります。

認証情報プロバイダーチェーン

すべての SDK には、 AWS 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。コード内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設定され、次に共有 AWS `config`ファイルで設定されます。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 config 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、 は認証情報プロバイダーを次の表に示す順序で AWS SDK for JavaScript 確認します。

AWS SDK for JavaScript API リファレンスの認証情報プロバイダーの優先順位によるメソッド	使用可能な認証情報プロバイダー	AWS SDKs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 `config`および `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 SDK for JavaScript API リファレンスの認証情報プロバイダーの優先順位によるメソッド	使用可能な認証情報プロバイダー	AWS SDKsリファレンスガイド
	AWS IAM Identity Center 認証情報	IAM Identity Center 認証情報プロバイダー
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推奨方法は、Amazon Cognito ID 認証情報クライアント を使用することです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 ですが JavaScript、Node.js AWS SDK for JavaScript で を使用すると、ブラウザスクリプトSDKで を使用すること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、一部の を正常に使用するには、モジュールなどの他の Node.js モジュールのインポートや使用など、一般的な Node.js コーディングパターンに精通していることAPIsが不可欠ですFile System (fs)。

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。例えば、組み込みHTTPモジュールを含めるには、以下を使用します。

```
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。例えば、HTML ファイルを読み取ります。

```
// 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。また、ここで で使用できる追加の Node.js GitHubパッケージについても確認できます。

Node.js maxSockets で 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、使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
  });
});
```

SDK の値またはAgentオブジェクトを指定しない場合、 の maxSockets は 50 の値 JavaScript 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 でキープアライブ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、新しいリクエストごとに新しい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、存続期間の短いオペレーションの場合、TCP接続を設定する際のレイテンシーオーバーヘッドは、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、次の DynamoDB クライアントの例に示すように、サービスごとのクライアントベースkeepAliveで でこれらの接続の再利用を無効にすることができます。

```
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は、 で Keep-Alive TCP パケットの初期遅延を設定することもできます。デフォルトでは keepAliveMsecs1000 ミリ秒です。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 はサードパーティーHTTPエージェントを介したHTTPまたは HTTPS プロキシの使用 JavaScript をサポートします。

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

サードパーティー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。*PROXY* 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

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

for Browsers SDK の構築

SDK JavaScript バージョン 2 (V2) とは異なり、V3 はデフォルトのサービスセットをサポートする JavaScript ファイルとして提供されません。代わりに V3 を使用すると、必要なSDK JavaScript ファイルの のみをバンドルしてブラウザに含めることができるため、オーバーヘッドが軽減されます。Webpack を使用して、 JavaScript ファイルSDKに必要な と、必要な追加のサードパーティーパッケージを 1 つのJavascriptファイルにバンドルし、<script>タグを使用してブラウザスクリプトにロードすることをお勧めします。Webpackの詳細については、「Webpack でアプリケーションをバンドルする」を参照してください。Webpack を使用して の V3 をブラウザ JavaScript にロードする例については、SDK「」を参照してくださいDynamoDB にデータを送信するアプリケーションを構築します。

ブラウザCORSで を強制する環境のSDK外部で作業し、 SDKの 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 JavaScript、リポジトリをクローンし、 のデフォルトのホストバージョンを構築するのと同じビルドツールを実行することで、 のカスタムコピーをSDKローカルで構築できますSDK。以下のセクションでは、追加の サービスとAPIバージョンSDKを使用して を構築する手順について説明します。

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

> **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、「Webpack でアプリケーションをバンドルする」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 セキュリティが強制されている場合は、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。

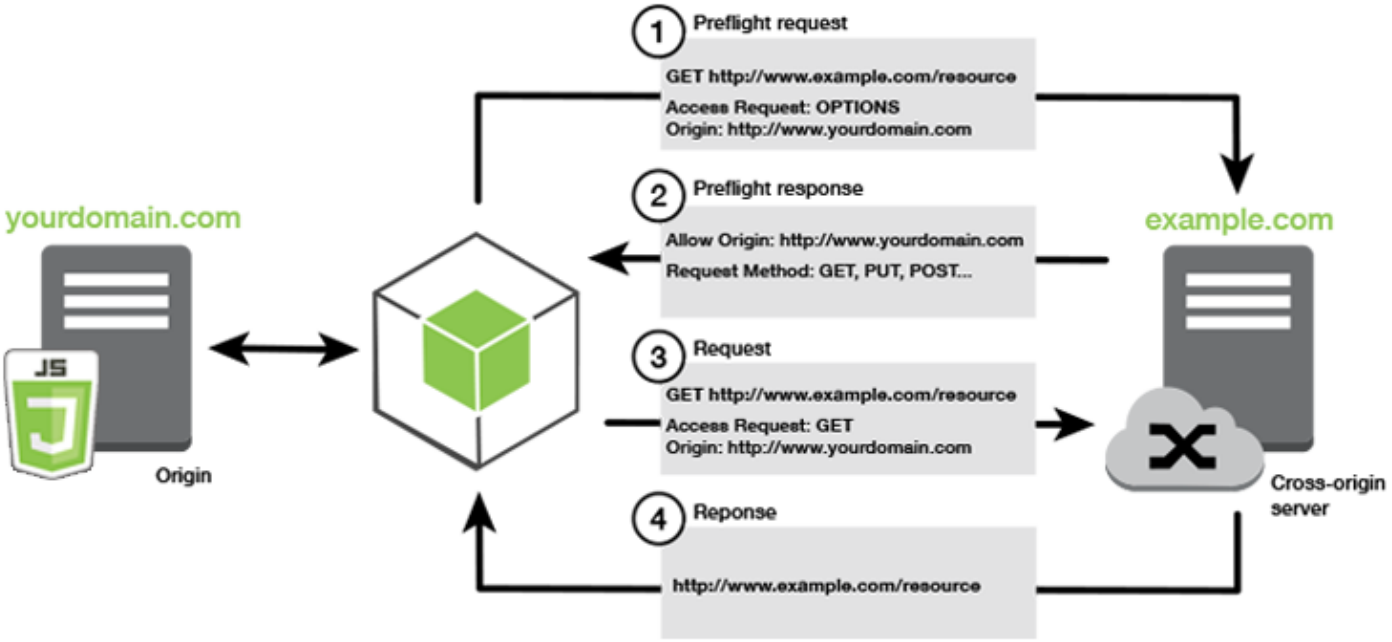
- リクエストを行う特定のドメイン
- 行われている HTTP リクエストのタイプ (GET、PUT、POST、DELETE など)

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 ウェブサービス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 ウェブサービス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て、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。

Webpack でアプリケーションをバンドルする

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。の SDK は、出力バンドルに含める依存関係の 1 つwebpackとして に含める JavaScript ことができます。

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

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。webpack CLI と 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 の設定

デフォルトでは、Webpack はプロジェクトのルートディレクトリwebpack.config.jsで 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 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、JavaScript アプリケーションの実行に必要なすべての が含まれます。エントリポイントで指定されたコードが SDK for 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 JavaScript、そのコードは設定で指定することなくバンドルされます。

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
```

次に、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"}]
  }
  **/
};
```

の AWS のサービスSDKの操作 JavaScript

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APIsための低レベルを提供する 1 つ以上のクライアントクラスがあります。例えば、Amazon DynamoDB APIsは DynamoDB クラスを通じて使用できます。

SDK の を通じて公開されるサービスは、リクエスト/レスポンスパターン JavaScript 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は、サービスのエンドポイントに HTTP/HTTPS リクエストを送信します。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、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リクエストとレスポンスのライフサイクル全体が含まれます。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、 オペレーションに関連するオブジェクトにカプセル化され、コールバック関数や JavaScript promise などのいくつかの手法のいずれかを使用してリクエスタに返されます。

トピック

- サービスオブジェクトを作成して呼び出す
- サービスを非同期的に呼び出す
- サービスクライアントリクエストを作成する
- サービスクライアントのレスポンスを処理する
- JSON を使用する
- SDK JavaScript コード例の

サービスオブジェクトを作成して呼び出す

は JavaScript API、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。の各 JavaScript サービスは、クライアントクラスに、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APIを提供します。のサービスクラス、オペレーション、パラメータの詳細については、「 APIリファレンス JavaScript API」を参照してください。

Node.js SDKで を使用する場合は、 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import、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ときは、 のJSON必要に応じて にパラメータを渡しますAPI。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 パラメータの受け渡しの詳細については、「」を参照してくださいJSON を使用する。

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

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

サービスを非同期的に呼び出す

を介して行われたすべてのリクエストSDKは非同期です。これは、ウェブブラウザでブラウザ scripts. JavaScript running を記述するときに、通常、実行スレッドが 1 つだけであることに留意することが重要です。 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できます。

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て、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

非同期呼び出しの管理

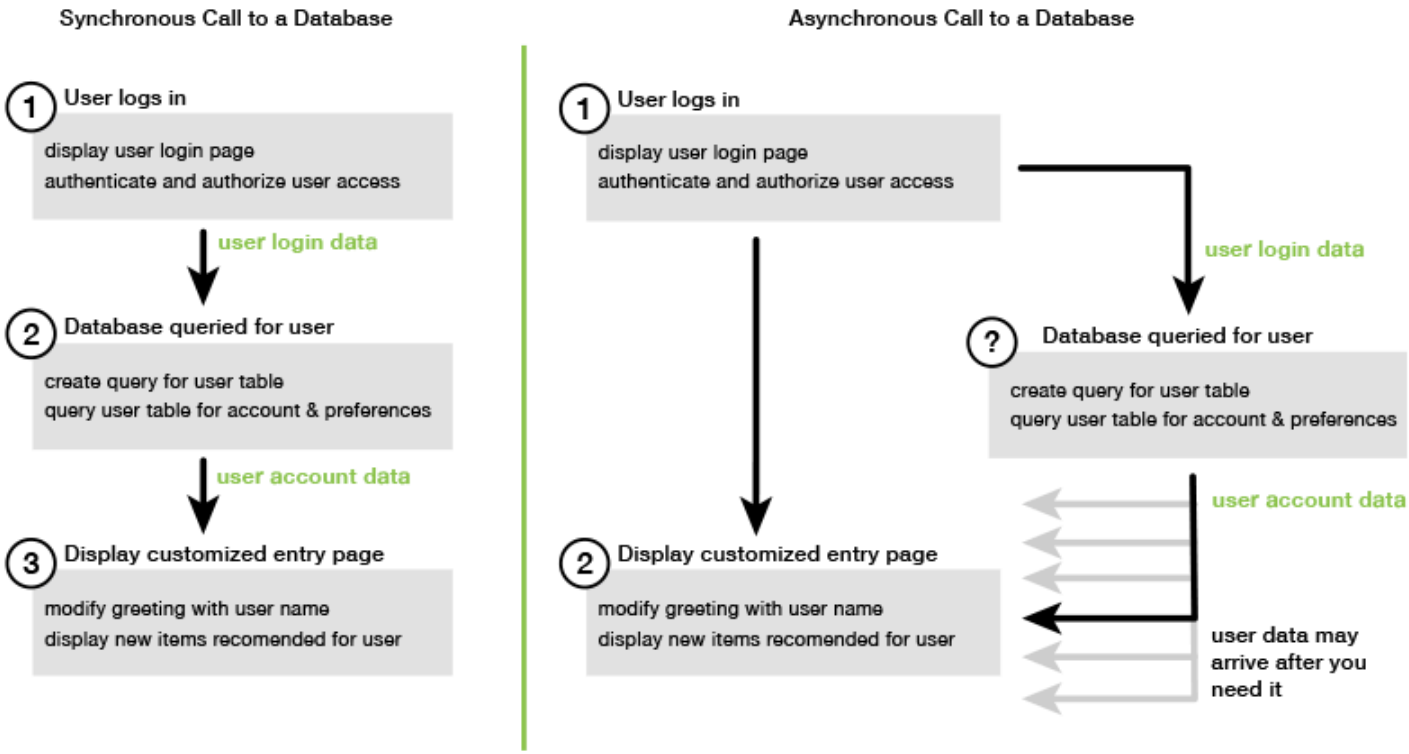
たとえば、e コマースウェブサイトのホームページは、リピート顧客がサインインするようにします。サインインする顧客にとっての利点の 1 つは、サインイン後に、顧客の特定の好みに合わせてサイトがカスタマイズされることです。これを実現するには、以下のことが必要です。

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 promise 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 メソッド () `ListTablesCommand` 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 プロミス (ECMAScript 2015) のサポートは、 JavaScriptコードが実行される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での JavaScript promise のサポートを判断するには、「」のECMAScript「 互換性表」を参照してください GitHub。

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 用の JavaScript のバージョン 3 (V3) では、リクエストを送信できます。

> **Note**
>
> SDK の の V3 を使用する場合、バージョン 2 (V2) コマンドを使用してオペレーションを実行することもできます JavaScript。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、リクエストの値を使用してリクエスト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のインターフェイスのAPIリファレンストピックを見ることで、リクエストのパラメータを調べることができます。例えば、*AbcCommand* クライアントメソッド、リクエストインターフェイスは *AbcInput*.
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、レスポンスオブジェクトは です *AbcResponse* (インターフェイス) タイプ。

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

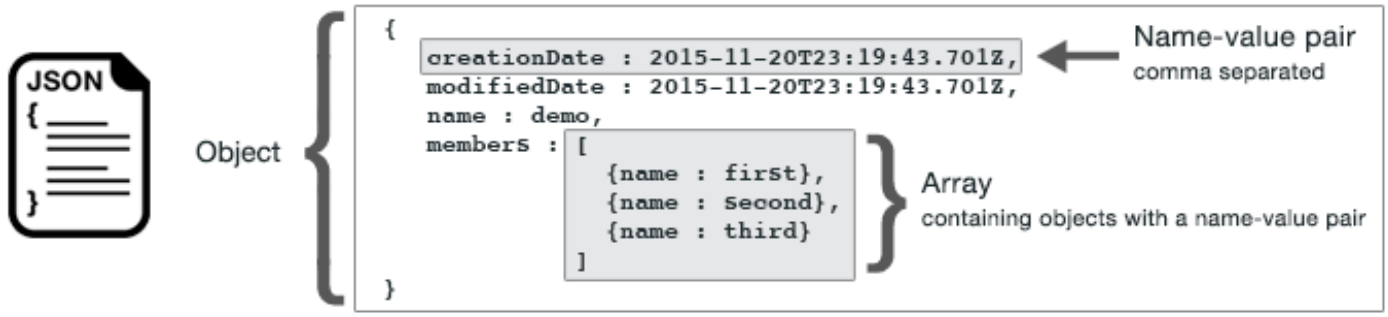
コマンドが失敗した場合、例外が発生します。次のコードスニペットは、サービス例外を処理する方法を示しています。

```
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 を使用する

JSON は、人間が読める形式と機械が読める形式の両方であるデータ交換用の形式です。名前JSONは JavaScript Object Notation の頭字語ですが、 の形式JSONはプログラミング言語から独立しています。

AWS SDK for JavaScript は JSON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 として受信しますJSON。の詳細についてはJSON、json.org を参照してください。



JSON は、次の 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。

以下は、JSONオブジェクトがカードゲームのカードを表すオブジェクトの配列を含むオブジェクトの例です。各カードは 2 つの名前と値のペアで定義されます。1 つはカードを識別する一意の値を指定し、もう 1 つは対応するカードイメージURLを指す 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"}
];
```

JSON as サービスオブジェクトパラメータ

以下は、 AWS Lambda サービスオブジェクトへの呼び出しのパラメータを定義するJSONために使用されるシンプルな 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メソッド呼び出しにパラメータを渡すときは、次の Lambda 関数を呼び出す例に示すように、メソッド呼び出しに JSON オブジェクトを指定します。

```
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 };
};
```

SDK JavaScript コード例の

このセクションのトピックでは、APIsさまざまな サービスの 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を示します。

これらの例のソースコードなどについては、 のAWS 「コード例リポジトリ」を参照してくださいGitHub。 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、より広範なシナリオとユースケースをカバーするコード例と、個々のAPI呼び出しのみをカバーするシンプルなコードス二ペットを作成しようとしています。手順については、 の寄稿ガイドライン GitHub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インポート/エクスポート構文を使用しています。
>
> - 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();
```

Amazon DynamoDB の例

Amazon DynamoDB は、ドキュメントストアモデルとキーバリューストアモデルの両方をサポートするフルマネージドの NoSQL クラウドデータベースです。専用データベースサーバーをプロビジョニングまたは保守する必要はなく、データ用のスキーマレステーブルを作成できます。

for JavaScript APIDynamoDB はDynamoDB、、DynamoDBStreams、および DynamoDB.DocumentClientクライアントクラスを通じて公開されます。DynamoDB クライアントクラスの使用の詳細については、 APIリファレンスの「Class: DynamoDB 」、「Class: DynamoDBStreams」、および「Class: DynamoDB ユーティリティ」を参照してください。

トピック

- DynamoDB の表の作成と使用
- DynamoDB での単一項目の読み取りと書き込み
- DynamoDB のバッチの項目の読み取りと書き込み
- DynamoDB 表のクエリおよびスキャニング
- DynamoDB ドキュメントクライアントの使用

DynamoDB の表の作成と使用

この Node.js コード例は以下を示しています。

- DynamoDB からデータを保存および取得するために使用されるテーブルを作成および管理する方法。

シナリオ

他のデータベース管理システムと同様、DynamoDB はデータをテーブルに保存します。DynamoDB テーブルは、行に相当する項目に整理されたデータの集まりです。DynamoDB, にデータを保存またはアクセスするには、テーブルを作成して使用します。

この例では、一連の Node.js モジュールを使用して DynamoDB テーブルで基本的な操作を実行します。このコードは、 SDKの JavaScript を使用して、DynamoDBクライアントクラスの次のメソッドを使用してテーブルを作成し、操作します。

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

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.js 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モジュールをインストールします。「」の指示に従ってください GitHub。
- for JavaScript DynamoDB クライアントSDKをインストールします。詳細については、「バージョン 3 の新機能とは」を参照してください。
- 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、「」および「ツールリファレンスガイド」の「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。 AWS SDKs

> **Important**
>
> これらの例では ECMAScript6 () を使用していますES6。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。

> **Note**
>
> これらの例で使用されているデータ型の詳細については、「Amazon DynamoDB でサポートされるデータ型と命名規則」を参照してください。

テーブルの作成

`create-table.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。テーブルの作成に必要なパラメータを含むJSONオブジェクトを作成します。この例には、各属性の名前とデータ型、キースキーマ、テーブルの名前、プロビジョニングするスループットの単位が含まれます。DynamoDB サービスオブジェクトの `CreateTableCommand` メソッドを呼び出します。

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

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

```
export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

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

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-table.js
```

このサンプルコードは、 にあります GitHub。

表の一覧を表示します

list-tables.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。テーブルの一覧表示に必

要なパラメータを含むJSONオブジェクトを作成します。この例では、一覧表示されるテーブルの数を 10 に制限します。DynamoDB サービスオブジェクトの ListTablesCommand メソッドを呼び出します。

```
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;
};
```

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-tables.js
```

このサンプルコードは、 にあります GitHub。

表の説明

describe-table.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。DynamoDB サービスJSONオブジェクトのDescribeTableCommandメソッドを記述するために必要なパラメータを含むオブジェクトを作成します。

```
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;
};
```

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 describe-table.js
```

このサンプルコードは、 にあります GitHub。

テーブルの削除

delete-table.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。テーブルの削除に必要なパラメータを含むJSONオブジェクトを作成します。この例には、コマンドラインパラメータとして指定されたテーブルの名前が含まれます。DynamoDB サービスオブジェクトの DeleteTableCommand メソッドを呼び出します。

```
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;
};
```

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-table.js
```

このサンプルコードは、 にあります GitHub。

DynamoDB での単一項目の読み取りと書き込み

この Node.js コード例は以下を示しています。

- DynamoDB テーブルに項目を追加する方法。
- DynamoDB テーブルで項目を取得する方法。
- DynamoDB テーブルで項目を削除する方法。

シナリオ

この例では、DynamoDB クライアントクラスのこれらのメソッドを使用して、一連の Node.js モジュールで、DynamoDB テーブル内の 1 つの項目を読み書きします。

- PutItemCommand
- UpdateItemCommand
- GetItemCommand
- DeleteItemCommand

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.js 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モジュールをインストールします。「」の指示に従ってください GitHub。
- 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、「」および「ツールリファレンスガイド」の「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。 AWS SDKs
- 項目にアクセスできる DynamoDB テーブルを作成します。DynamoDB テーブルを作成する方法の詳細については、DynamoDB の表の作成と使用 を参照してください。

> **Important**
>
> これらの例では ECMAScript6 () を使用していますES6。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。

> **Note**
>
> これらの例で使用されているデータ型の詳細については、「Amazon DynamoDB でサポートされるデータ型と命名規則」を参照してください。

項目を書き込みます

put-item.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。項目の追加に必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前と、設定する属性と各属性の値を定義するマップが含まれます。DynamoDB クライアントサービスオブジェクトの PutItemCommand メソッドを呼び出します。

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new PutItemCommand({
    TableName: "Cookies",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Item: {
      Flavor: { S: "Chocolate Chip" },
      Variants: { SS: ["White Chocolate Chip", "Chocolate Chunk"] },
    },
```

```
  });

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

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 put-item.js
```

このサンプルコードは、 にあります GitHub。

項目を更新する

update-item.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。項目の追加に必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前、更新するキー、新しい属性名をマッピングする日付式、および新しい各属性の値が含まれます。DynamoDB クライアントサービスオブジェクトの UpdateItemCommand メソッドを呼び出します。

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

const client = new DynamoDBClient({});

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

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

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

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 update-item.js
```

このサンプルコードは、 にあります GitHub。

項目の取得

get-item.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDBクライアントサービスオブジェクトを作成します。取得する項目を特定するには、テーブルのその項目のプライマリキーの値を指定する必要があります。デフォルトでは、GetItemCommand メソッドは項目に定義されているすべての属性値を返します。使用可能なすべての属性値のサブセットのみを取得するには、プロジェクション式を指定します。

項目の取得に必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前、取得する項目の名前と値、取得する項目属性を識別する射影式が含まれます。DynamoDB クライアントサービスオブジェクトの GetItemCommand メソッドを呼び出します。

次のコード例では、パーティションキーとソートキーの両方ではなく、パーティションキーのみで構成されるプライマリキーを持つテーブルから項目を取得します。表にパーティションキーとソートキーで構成されるプライマリキーがある場合は、ソートキーの名前と属性も指定する必要があります。

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new GetItemCommand({
    TableName: "CafeTreats",
    // For more information about data types,
```

```
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      TreatId: { N: "101" },
    },
  });

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

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-item.js
```

このサンプルコードは、 にあります GitHub。

項目の削除

delete-item.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。項目の削除に必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前と、削除する項目のキー名と値の両方が含まれます。DynamoDB クライアントサービスオブジェクトの DeleteItemCommand メソッドを呼び出します。

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteItemCommand({
    TableName: "Drinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
```

```
      Name: { S: "Pumpkin Spice Latte" },
    },
  });

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

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-item.js
```

このサンプルコードは、 にあります GitHub。

DynamoDB のバッチの項目の読み取りと書き込み

この Node.js コード例は以下を示しています。

- DynamoDB テーブルの項目のバッチを読み書きする方法。

シナリオ

この例では、一連のNode.jsモジュールを使用して、アイテムのバッチをDynamoDB表に配置し、アイテムのバッチを読み取ります。このコードは、 SDKの JavaScript を使用して、DynamoDB クライアントクラスの次のメソッドを使用してバッチ読み取りおよび書き込みオペレーションを実行します。

- BatchGetItemCommand
- BatchWriteItemCommand

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。

- これらのノード TypeScript 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モジュールをインストールします。「」の指示に従ってください GitHub。
- 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、「」および「ツールリファレンスガイド」の「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。 AWS SDKs
- 項目にアクセスできる DynamoDB テーブルを作成します。DynamoDB テーブルを作成する方法の詳細については、DynamoDB の表の作成と使用 を参照してください。

> **Important**
>
> これらの例では ECMAScript6 () を使用していますES6。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。

> **Note**
>
> これらの例で使用されているデータ型の詳細については、「Amazon DynamoDB でサポートされるデータ型と命名規則」を参照してください。

項目の一括読み取り

`batch-get-item.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。項目のバッチを取得するために必要なパラメータを含むJSONオブジェクトを作成します。この例には、読み取る 1 つ以上のテーブルの名前、各テーブルで読み取るキーの値、および返す属性を指定する射影式が含まれます。DynamoDB サービスオブジェクトの `BatchGetItemCommand` メソッドを呼び出します。

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

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

```
export const main = async () => {
  const command = new BatchGetItemCommand({
    RequestItems: {
      // Each key in this object is the name of a table. This example refers
      // to a PageAnalytics table.
      PageAnalytics: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            // "PageName" is the partition key (simple primary key).
            // "S" specifies a string as the data type for the value "Home".
            // For more information about data types,
            // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
            // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
            PageName: { S: "Home" },
          },
          {
            PageName: { S: "About" },
          },
        ],
        // Only return the "PageName" and "PageViews" attributes.
        ProjectionExpression: "PageName, PageViews",
      },
    },
  });

  const response = await client.send(command);
  console.log(response.Responses["PageAnalytics"]);
  return response;
};
```

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 batch-get-item.js
```

このサンプルコードは、 [にあります GitHub]。

項目の一括書き込み

batch-write-item.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にア

クセスするには、DynamoDB クライアントサービスオブジェクトを作成します。項目のバッチを取得するために必要なパラメータを含むJSONオブジェクトを作成します。この例には、項目を書き込むテーブル、各項目に書き込むキー、および属性とその値が含まれます。DynamoDB サービスオブジェクトの BatchWriteItemCommand メソッドを呼び出します。

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

const client = new DynamoDBClient({});

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

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

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

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 batch-write-item.js
```

このサンプルコードは、 にあります GitHub。

DynamoDB 表のクエリおよびスキャニング

この Node.js コード例は以下を示しています。

- 項目の DynamoDB テーブルをクエリおよびスキャンする方法。

シナリオ

クエリでは、プライマリキーの属性値のみを使用してテーブルまたはセカンダリインデックスの項目を検索します。パーティションキーの名前と検索対象の値を指定する必要があります。ソートキーの名前と値を指定し、比較演算子を使用して、検索結果をさらに絞り込むこともできます。スキャンは、指定したテーブルのすべての項目をチェックして項目を探します。

この例では、一連の Node.js モジュールを使用して、DynamoDB テーブルから取得する 1 つ以上の項目を識別します。このコードは、 SDKの JavaScript を使用して、DynamoDB クライアントクラスの次のメソッドを使用してテーブルをクエリおよびスキャンします。

- QueryCommand
- ScanCommand

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.js 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モジュールをインストールします。「」の指示に従ってください GitHub。
- 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、「」および「ツールリファレンスガイド」の「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。 AWS SDKs
- 項目にアクセスできる DynamoDB テーブルを作成します。DynamoDB テーブルを作成する方法の詳細については、DynamoDB の表の作成と使用 を参照してください。

> **Important**
>
> これらの例では ECMAScript6 () を使用していますES6。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。
>
> > **Note**
> >
> > これらの例で使用されているデータ型の詳細については、「Amazon DynamoDB でサポートされるデータ型と命名規則」を参照してください。

テーブルに対するクエリの実行

`query.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。テーブルのクエリに必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブル名、クエリ`ExpressionAttributeValues`に必要な 、それらの値`KeyConditionExpression`を使用してクエリが返す項目を定義する 、および各項目に対して返す属性値の名前が含まれます。DynamoDB サービスオブジェクトの `QueryCommand` メソッドを呼び出します。

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new QueryCommand({
    KeyConditionExpression: "Flavor = :flavor",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":flavor": { S: "Key Lime" },
      ":searchKey": { S: "no coloring" },
    },
    FilterExpression: "contains (Description, :searchKey)",
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

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 query.js
```

このサンプルコードは、 にあります GitHub。

表のスキャン

scan.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ダウンロードを含め、前述のSDKように を設定してください。DynamoDB にアクセスするには、DynamoDB クライアントサービスオブジェクトを作成します。テーブルで項目をスキャンするために必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前、一致する各項目に対して返される属性値のリスト、および指定されたフレーズを含む項目を見つけ

るために結果セットをフィルタリングする式が含まれます。DynamoDB サービスオブジェクトの ScanCommand メソッドを呼び出します。

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ScanCommand({
    FilterExpression: "CrustType = :crustType",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":crustType": { S: "Graham Cracker" },
    },
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

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 scan.js
```

このサンプルコードは、 にあります GitHub。

DynamoDB ドキュメントクライアントの使用

この Node.js コード例は以下を示しています。

- DynamoDB ユーティリティを使用して DynamoDB 表にアクセスする方法。

シナリオ

DynamoDB ドキュメントクライアントは、属性値の概念を抽象化することによって項目の操作を簡単にします。この抽象化は、入力パラメータとして提供されるネイティブ JavaScript 型に注釈を付け、注釈付きレスポンスデータをネイティブ JavaScript型に変換します。

DynamoDB ドキュメントクライアントの詳細については、「」の「@aws-sdk/lib-dynamodbREADME」を参照してください GitHub。Amazon DynamoDB を使用したプログラミングの詳細については、「Amazon DynamoDB デベロッパーガイド」の「 を使用した JavaScript Amazon DynamoDB のプログラミング」を参照してください。

この例では、一連の Node.js モジュールを使用して、DynamoDB ユーティリティを使用するDynamoDB表で基本操作を実行します。このコードは、 SDKの JavaScript を使用して、DynamoDB ドキュメントクライアントクラスの次のメソッドを使用してテーブルをクエリおよびスキャンします。

- GetCommand
- PutCommand
- UpdateCommand
- QueryCommand
- DeleteCommand

DynamoDB ドキュメントクライアントの設定の詳細については、@aws-sdk/lib-dynamodb 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します。

- これらの Node.js 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モジュールをインストールします。「」の指示に従ってください GitHub。
- 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、「」および「ツールリファレンスガイド」の「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。 AWS SDKs

- 項目にアクセスできる DynamoDB テーブルを作成します。の を使用して DynamoDB テーブルを作成する方法の詳細については JavaScript、SDK「」を参照してくださいDynamoDB の表の作成と使用。DynamoDB コンソールを使用してテーブルを作成することもできます。

> **Important**
>
> これらの例では ECMAScript6 () を使用していますES6。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。

> **Note**
>
> これらの例で使用されているデータ型の詳細については、「Amazon DynamoDB でサポートされるデータ型と命名規則」を参照してください。

テーブルからの項目の取得

get.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SDKように を設定してください。これはドキュメントクライアント機能を @aws-sdk/client-dynamodb に提供する @aws-sdk/lib-dynamodb のライブラリパッケージを含みます。次に、ドキュメントクライアントの作成時に、オプションの 2 番目のパラメータとしてマーシャリングとアンマーシャリングの設定を以下に示すように設定します。次に、クライアントを作成します。次に、テーブルから項目を取得するのに必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前、そのテーブル内のハッシュキーの名前、取得する項目のハッシュキーの値が含まれます。DynamoDB ドキュメントクライアントの GetCommand メソッドを呼び出します。

```
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;
};
```

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.js
```

このサンプルコードは、 にあります GitHub。

テーブルでの項目の入力

put.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SDKように を設定してください。これはドキュメントクライアント機能を @aws-sdk/client-dynamodb に提供する @aws-sdk/lib-dynamodb のライブラリパッケージを含みます。次に、ドキュメントクライアントの作成時に、オプションの 2 番目のパラメータとしてマーシャリングとアンマーシャリングの設定を以下に示すように設定します。次に、クライアントを作成します。テーブルに項目を書き込むために必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前と、項目に追加または更新する項目の説明が含まれます。これには、項目に設定する属性のハッシュキー、値、名前、値が含まれます。DynamoDB ドキュメントクライアントの PutCommand メソッドを呼び出します。

```
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;
};
```

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 put.js
```

このサンプルコードは、 にあります GitHub。

テーブルでの項目の更新

update.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SDKように を設定してください。これはドキュメントクライアント機能を @aws-sdk/client-dynamodb に提供する @aws-sdk/lib-dynamodb のライブラリパッケージを含みます。次に、ドキュメントクライアントの作成時に、オプションの 2 番目のパラメータとしてマーシャリングとアンマーシャリングの設定を以下に示すように設定します。次に、クライアントを作成します。テーブルに項目を書き込むために必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前、更新する項目のキー、ExpressionAttributeValuesパラメータで値を割り当てるトークンで更新UpdateExpressionsする項目の属性を定義する のセットが含まれます。DynamoDB ドキュメントクライアントの UpdateCommandメソッドを呼び出します。

```
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;
};
```

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 update.js
```

このサンプルコードは、 にあります GitHub。

テーブルに対するクエリの実行

query.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SDKように を設定してください。これはドキュメントクライアント機能を @aws-sdk/client-dynamodb に提供する @aws-sdk/lib-dynamodb のライブラリパッケージを含みます。テーブルのクエリに必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブル名、クエリExpressionAttributeValuesに必要な 、およびそれらの値を使用してクエリKeyConditionExpressionが返す項目を定義する が含まれます。DynamoDB ドキュメントクライアントの QueryCommand メソッドを呼び出します。

```
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;
};
```

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 query.js
```

このサンプルコードは、 にあります GitHub。

テーブルからの項目の削除

delete.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SDKように を設定してください。これはドキュメントクライアント機能を @aws-sdk/client-dynamodb に提供する @aws-sdk/lib-dynamodb のライブラリパッケージを含みます。次に、ドキュメントクライアントの作成時に、オプションの 2 番目のパラメータとしてマーシャリングとアンマーシャリングの設定を以下に示すように設定します。次に、クライアントを作成します。DynamoDB にアクセスするには、DynamoDBオブジェクトを作成します。テーブル内の項目を削除するために必要なパラメータを含むJSONオブジェクトを作成します。この例には、テーブルの名前と、削除する項目のハッシュキーの名前と値が含まれます。DynamoDB ドキュメントクライアントの DeleteCommand メソッドを呼び出します。

```
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;
```

```
};
```

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.js
```

このサンプルコードは、 にあります GitHub。

AWS Elemental MediaConvert の例

AWS Elemental MediaConvert は、ブロードキャストグレードの機能を備えたファイルベースの動画変換サービスです。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ユーザーガイドを参照してください。

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 のリージョン固有のエンドポイントの取得
- 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 のリージョン固有のエンドポイントの取得

この Node.js コード例は以下を示しています。

- MediaConvert からリージョン固有のエンドポイントを取得する方法。

シナリオ

次の例では、Node.js モジュールを使用して MediaConvert を呼び出し、リージョン固有のエンドポイントを取得します。エンドポイント URL はサービスのデフォルトエンドポイントから取得できる

ため、リージョン固有のエンドポイントはまだ必要ありません。コードは SDK for JavaScript を使用して、MediaConvert クライアントクラスのこのメソッドを使用してこのエンドポイントを取得します。

- `DescribeEndpointsCommand`

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設定し、必要なAWS SDK for JavaScriptとサードパーティーの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。

エンドポイント URL を取得

`libs`ディレクトリを作成し、ファイル名`emcClientGet.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`* (地域) を、AWS地域に置き換えます。

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

```
const REGION = "REGION";
//Set the MediaConvert Service Object
const emcClientGet = new MediaConvertClient({ region: REGION });
export { emcClientGet };
```

このサンプルコードは、このGitHubにで見つけられます。

emc_getendpoint.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 DescribeEndpointsCommand 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。次に、DescribeEndpointsCommand メソッドを呼び出します。

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

//set the parameters.
const params = { MaxResults: 0 };

const run = async () => {
  try {
    // Create a new service object and set MediaConvert to customer endpoint
    const data = await emcClientGet.send(new DescribeEndpointsCommand(params));
    console.log("Your MediaConvert endpoint is ", data.Endpoints);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

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_getendpoint.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設定し、必要なAWS SDK for JavaScriptとサードパーティーの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*（地域）を、AWS地域に置き換えます。*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*（地域）を、AWS地域に置き換えます。*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* (地域) を、AWS地域に置き換えます。*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設定し、必要なAWS SDK for JavaScriptとサードパーティーの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`*（地域）を、AWS地域に置き換えます。*`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*（地域）を、AWS地域に置き換えます。*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* (地域) を、AWS地域に置き換えます。*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*（地域）を、AWS地域に置き換えます。*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 の例

AWS Lambdaはサーバーレスコンピューティングサービスであり、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ター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ます。

AWS Lambda 向けJavaScript API は LambdaService （Lambdaサービス）クライアントクラスを介して公開されます。

v3のAWS SDK for JavaScript Lambda関数を作成して使用する方法を示す例の一覧表を次に示します。

- 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を作成する

Amazon Lex での例

Amazon Lex は、アプリケーションに音声とテキストによる会話型インターフェイスを構築するための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。

- 「」の手順に従って、Node JavaScript サンプルを実行するようにプロジェクト環境を設定します GitHub。
- 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 permissions、および 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。

```
{
  "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 では、 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。の詳細については 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 「 AWS CLI ユーザーガイド」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`setup.yaml`に という名前のファイルを作成し、そこにコンテンツをコピー GitHubします。

   > **Note**
   >
   > AWS CloudFormation テンプレートは、 AWS CDK で利用可能な GitHub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。

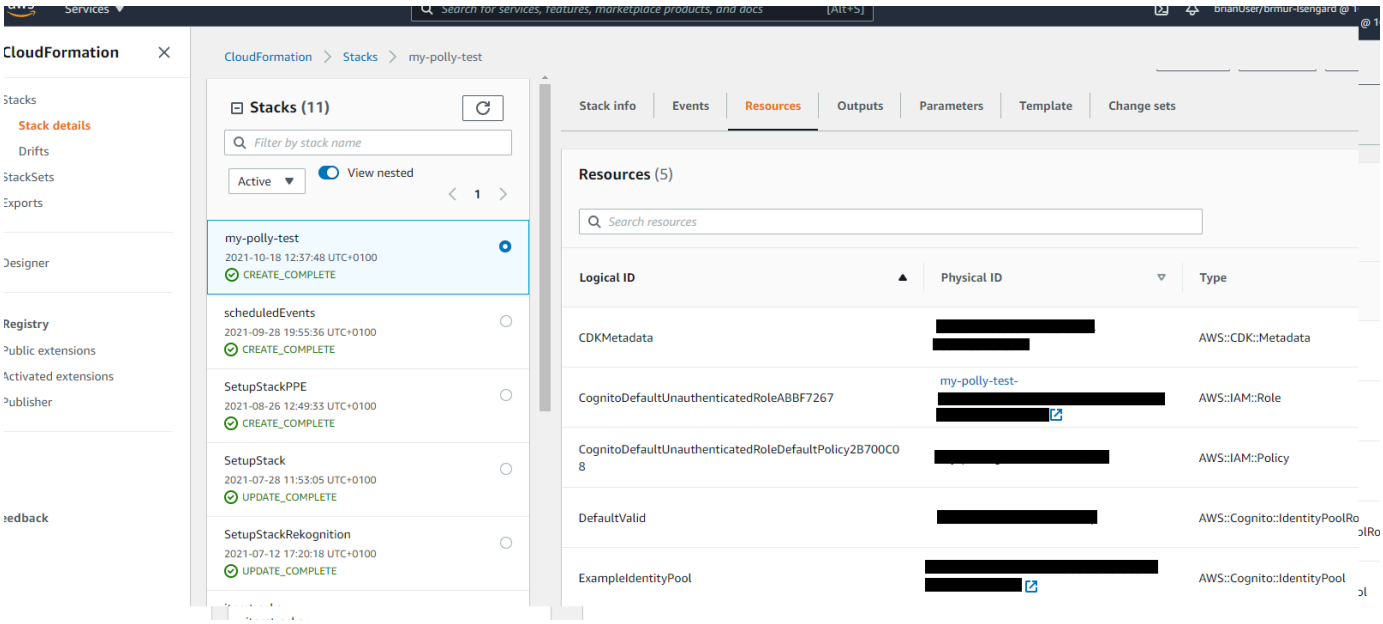
3. コマンドラインから以下のコマンドを実行し、「*`STACK_NAME`*」をスタックの一意の名前に置き換えます。

   > **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。

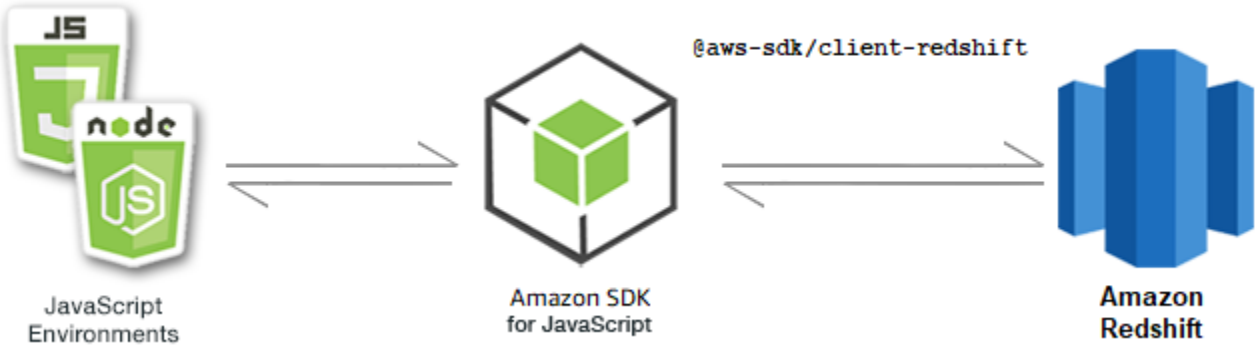
```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "./libs/pollyClient.js";

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

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

このサンプルコードは、 にあります GitHub。

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設定し、必要なAWS SDK for JavaScriptとサードパーティーの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* (地域) を、AWS地域に置き換えます。

```
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 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。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*（地域）を、AWS地域に置き換えます。

```
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 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、変更したいクラスターの名前、および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*（地域）を、AWS地域に置き換えます。

```
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 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、変更したいクラスターの名前、および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* (地域) を、AWS地域に置き換えます。

```
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、変更したいクラスターの名前、および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();
```

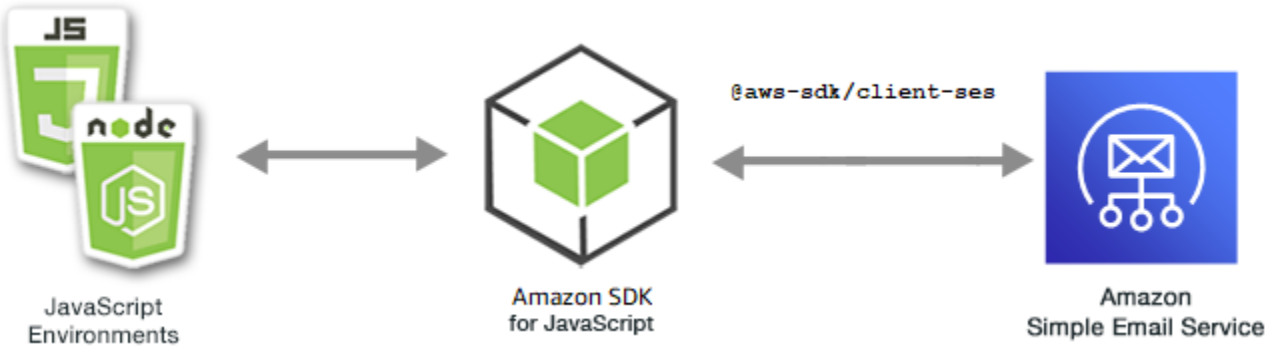
この例を実行するには、コマンドプロンプトで以下を入力します。

```
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 E メールを送信するのに役立つ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。



for Amazon JavaScript APISES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の詳細については、 APIリファレンスの「Class:SES」を参照してください。

トピック

- Amazon ID SES の管理
- Amazon での E メールテンプレートの使用 SES
- Amazon を使用した E メールの送信 SES

Amazon ID SES の管理

この Node.js コード例は以下を示しています。

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

- Amazon で使用される ID を削除する方法SES。

Amazon SES ID は、Amazon が E メールを送信SESするために使用する 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 ID を検証し、自分が所有していることを確認し、他のユーザーによる使用を防止する必要があります。

Amazon で E メールアドレスとドメインを検証する方法の詳細についてはSES、Amazon Simple Email Service デベロッパーガイドの「Amazon での E メールアドレスとドメインSESの検証」を参照してください。Amazon での送信承認の詳細についてはSES、「Amazon SES送信承認の概要」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 ID SES を検証および管理します。Node.js モジュールは、 SDKの JavaScript を使用して、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。

- これらのノード TypeScript 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モジュールをインストールします。「」の指示に従ってください GitHub。
- 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、「」および「ツールリファレンスガイド」の「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。 AWS SDKs

> **Important**
>
> これらの例は、 ECMAScript6 () 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インポート/エクスポートする方法を示していますES6。

- 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 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SES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように 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呼び出し、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**
>
> 置換 *IdentityType* ID タイプはEmailAddress「」または「ドメイン」です。

```
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、 にあります GitHub。

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 で使用する E メール送信者を検証します SES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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。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、前述のSDKように 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、パラメータを渡して Amazon SESクライアントサービスオブジェクトを呼び出します。

> **Note**
>
> 置換 *EMAIL_ADDRESS* name@example.com などの E メールアドレス。

```
// 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。ドメインが Amazon に追加され SES、検証されます。

```
node ses_verifyemailidentity.js
```

このサンプルコードは、 にあります GitHub。

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 で使用する E メールドメインを検証します SES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように 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呼び出し、パラメータオブジェクトを渡します。

> **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**
>
> 置換 *DOMAIN_NAME* ドメイン名の 。

```
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。ドメインが Amazon に追加され SES、検証されます。

```
node ses_verifydomainidentity.js
```

このサンプルコードは、 にあります GitHub。

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 で使用される E メールアドレスまたはドメインを削除しますSES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように 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呼び出すrequestための 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**
>
> 置換 *IDENTITY_EMAIL* 削除する ID の E メールを入力します。

```
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、 にあります GitHub。

Amazon での E メールテンプレートの使用 SES

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 で E メールテンプレートを作成して使用する方法の詳細については SES、Amazon SES Simple Email Service デベロッパーガイドの「Amazon を使用したパーソナライズされた E API 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の を使用して JavaScript 、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および使用します。

- `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。

- これらのノード TypeScript 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モジュールをインストールします。「」の指示に従ってください GitHub。

- 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、「」および「ツールリファレンスガイド」の「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。 AWS SDKs

> **Important**
>
> これらの例は、 ECMAScript6 () 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インポート/エクスポートする方法を示していますES6。
>
> - 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 で使用する E メールテンプレートを作成しますSES。

`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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *`REGION`* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように 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、パラメータを渡して Amazon SESクライアントサービスオブジェクトを呼び出します。

> **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 はテンプレートのリストSESを返します。

```
node ses_listtemplates.js
```

このサンプルコードは、 にあります GitHub。

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 で使用する E メールテンプレートを取得しますSES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように 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、パラメータを渡して Amazon SESクライアントサービスオブジェクトを呼び出します。

> **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**
>
> 置換 *TEMPLATE_NAME* 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 はテンプレートの詳細SESを返します。

```
node ses_gettemplate.js
```

このサンプルコードは、 にあります GitHub。

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 で使用する E メールテンプレートを作成しますSES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように 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、パラメータを渡して Amazon SESクライアントサービスオブジェクトを呼び出します。

> **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**
>
> 置換 *TEMPLATE_NAME* 新しいテンプレートの名前、*HtmlPart* E メールのHTMLタグ付けされたコンテンツ、および *SubjectPart* E メールの件名の 。

```
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 に追加されますSES。

```
node ses_createtemplate.js
```

このサンプルコードは、 にあります GitHub。

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 で使用する E メールテンプレートを作成しますSES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように 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、パラメータを渡して Amazon SESサービスオブジェクトを呼び出します。

> **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**
>
> 置換 *TEMPLATE_NAME* テンプレートの名前と *HTML_PART* E メールのHTMLタグ付けされたコンテンツを含む 。

```
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 はテンプレートの詳細SESを返します。

```
node ses_updatetemplate.js
```

このサンプルコードは、 にあります GitHub。

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 で使用する E メールテンプレートを作成しますSES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように 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、パラメータを渡して Amazon SESサービスオブジェクトを呼び出します。

> **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**
>
> 置換 *TEMPLATE_NAME* 削除するテンプレートの名前。

```
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 はテンプレートの詳細 SESを返します。

```
node ses_deletetemplate.js
```

このサンプルコードは、 にあります GitHub。

Amazon を使用した E メールの送信 SES

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 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 SESAPIでは、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未加工の 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 を使用してフォーマットされた E メールを送信するSESAPI」および「Amazon を使用して raw E SES メールを送信するAPI」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の を使用して JavaScript 、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および使用します。

- `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。

- これらのノード TypeScript 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モジュールをインストールします。「」の指示に従ってください GitHub。

- 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、「」および「ツールリファレンスガイド」の「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。 AWS SDKs

> **Important**
>
> これらの例は、 ECMAScript6 () 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インポート/エクスポートする方法を示していますES6。
>
> - 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、すぐに送信キューに入れます。`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 Mailbox Simulator 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。受信者アドレスには、宛先: アドレス、CC: アドレス、または BCC: アドレスを指定できます。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:、および : BCC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 で E メールを送信しますSES。

`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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *`REGION`* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように を設定します。

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

> **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**
>
> 置換 *toAddress* E メールを送信するアドレス、および *fromAddress* E メールの送信元の E メールアドレス。

```
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 によって送信用にキューに入れられますSES。

```
node ses_sendemail.js
```

このサンプルコードは、 にあります GitHub。

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 で E メールを送信しますSES。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SDKように を設定します。

オブジェクトを作成して、送信者と受信者のアドレス、件名、プレーンテキストと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メソッドに渡します。SendTemplatedEmailCommand メソッドを呼び出すには、パラメータを渡して Amazon SESクライアントサービスオブジェクトを呼び出します。

> **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**
>
> 置換 *REGION* AWS リージョン、*USER* E メールの送信先となる名前と E メールアドレス、*VERIFIED_EMAIL* E メールの送信元の E メールアドレス、および *TEMPLATE_NAME* をテンプレートの名前で指定します。

```
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 によって送信用にキューに入れられますSES。

```
node ses_sendtemplatedemail.js
```

このサンプルコードは、 にあります GitHub。

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 で E メールを送信しますSES。

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て貼り付けると、Amazon SESクライアントオブジェクトが作成されます。置換 *REGION* を AWS リージョンで使用します。

```
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、 にあります GitHub。

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SDKように を設定します。

オブジェクトを作成して、送信者と受信者のアドレス、件名、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プレーンテキストとHTML形式でSESクライアントクラスの SendBulkTemplatedEmailCommandメソッドに渡します。SendBulkTemplatedEmailCommand メソッドを呼び出すには、パラメータを渡して Amazon SESサービスオブジェクトを呼び出します。

> **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**
>
> 置換 *USERS* E メールの送信先となる名前と E メールアドレス、*VERIFIED_EMAIL_1* E メールの送信元の E メールアドレス、および *TEMPLATE_NAME* をテンプレートの名前で指定します。

```
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 によって送信用にキューに入れられますSES。

```
node ses_sendbulktemplatedemail.js
```

このサンプルコードは、 にあります GitHub。

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（Web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、SMSAWS 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設定し、必要なAWS SDK for JavaScriptとサードパーティーの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*（地域）を、AWS地域に置き換えます。

```
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*（地域）を、AWS地域に置き換えます。

```
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* (地域) を、AWS地域に置き換えます。

```
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* (地域) を、AWS地域に置き換えます。

```
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* (地域) を、AWS地域に置き換えます。

```
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設定し、必要なAWS SDK for JavaScriptとサードパーティーの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*（地域）を、AWS地域に置き換えます。

```
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設定し、必要なAWS SDK for JavaScriptとサードパーティーの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* (地域) を、AWS地域に置き換えます。

```
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* (地域) を、AWS地域に置き換えます。

```
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* (地域) を、AWS地域に置き換えます。

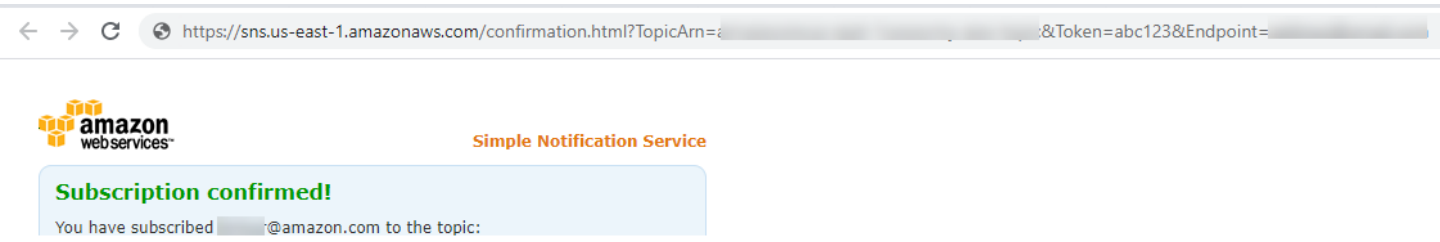
```
import { SNSClient } from "@aws-sdk/client-sns";

// 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* (地域) を、AWS地域に置き換えます。

```
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け取るようにします。

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* (地域) を、AWS地域に置き換えます。

```
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のTopicArn、およびAWS Lambda関数のAmazon Resource Name 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* (地域) を、AWS地域に置き換えます。

```
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設定し、必要なAWS SDK for JavaScriptとサードパーティーの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`* (地域) を、AWS地域に置き換えます。

```
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* (地域) を、AWS地域に置き換えます。

```
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* (地域) を、AWS地域に置き換えます。

```
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*（地域）を、AWS地域に置き換えます。

```
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* (地域) を、AWS地域に置き換えます。

```
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 の JavaScript API は、 TranscribeService クライアントクラスを通じて公開されます。

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。

- これらのノード TypeScript 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モジュールをインストールします。「」の指示に従ってください GitHub。
- 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、「」を参照してくださいStartTranscriptionJobCommand。

`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、 にあります GitHub。

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、 にあります GitHub。

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、 にあります GitHub。

`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、 にあります GitHub。

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、 にあります GitHub。

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、 にあります GitHub。

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。

- これらのノード TypeScript 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モジュールをインストールします。「」の指示に従ってください GitHub。
- 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、MedicalTranscription「ジョブの開始」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、 にあります GitHub。

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、 にあります GitHub。

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、ListTranscriptionMedicalJobs「 コマンド」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、 にあります GitHub。

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、 にあります GitHub。

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、 にあります GitHub。

`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、 にあります GitHub。

Amazon EC2インスタンスでの Node.js のセットアップ

SDK の で Node.js を使用する一般的なシナリオ JavaScript は、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスで Node.js ウェブアプリケーションをセットアップして実行することです。このチュートリアルでは、Linux インスタンスを作成し、 を使用して接続しSSH、Node.js をインストールしてそのインスタンスで実行します。

前提条件

このチュートリアルでは、インターネットからアクセスでき、 を使用して接続できるパブリック DNS名を持つ Linux インスタンスを既に起動していることを前提としていますSSH。詳細について

は、「Amazon 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。 EC2

> **Important**
>
> 新しい Amazon インスタンスを起動するときは、Amazon Linux 2023 Amazon マシンイメージ (AMI) を使用します。 EC2

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 ユーザーガイド」の「Amazon でのセットアップEC2」を参照してください。 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. `ec2-user` を使用して Linux インスタンスに接続しますSSH。
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 リポジトリを参照してください。

   ```
   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て、nvm を使用してLTS最新バージョンの Node.js 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、ノードを再度インストールする必要があります。代わりに、次のトピックで説明するように、保持する設定が完了したら、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 マシンイメージの作成 (AMI)

Amazon EC2インスタンスに 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。を作成するAMIと、同じ Node.js 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AMIから を作成する方法の詳細については、「Amazon 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。 EC2

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) – 「」の「nvm repo 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。

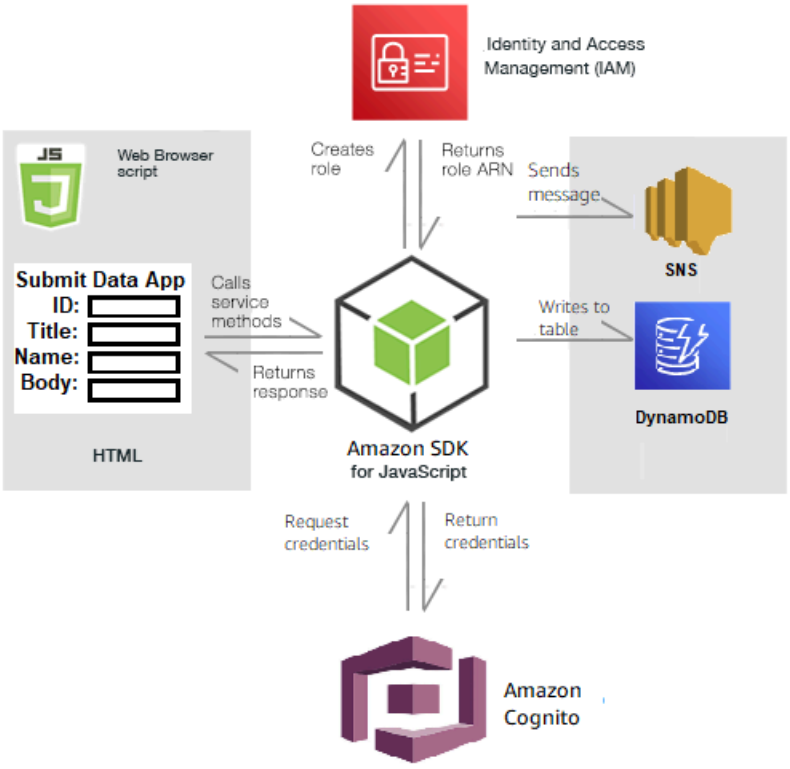
DynamoDB にデータを送信するアプリケーションを構築します

このクロスサービス Node.js チュートリアルでは、ユーザーが Amazon DynamoDB 表にデータの送信を有効にするアプリケーションを構築する方法について示します。このアプリケーションでは、次のサービスを使用します。

- AWS Identity and Access Management(IAM) と Amazon Cognito の認可と許可。
- 表を作成および更新するための「Amazon DynamoDB」（ Amazon DynamoDB ）。
- Amazon Simple Notification Service (Amazon SNS) は、ユーザーが表を更新したときにアプリケーション管理者に通知します。

シナリオ

このチュートリアルでは、HTML ページが Amazon DynamoDB表にデータを送信するためのブラウザベースのアプリケーションを提供します。アプリケーションは Amazon SNS を使用して、ユーザーが表を更新したときにアプリケーション管理者に通知します。



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. HTMLを作成する
4. ブラウザスクリプトを作成
5. 次のステップ

前提条件

以下の前提条件を満たしてください。

- 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とサードパーティーの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

このアプリケーションは、次のリソースが必要になります。

- 次の権限を持つ AWS Identity and Access Management (IAM) 認証されていない Amazon Cognito ユーザーロール
  - sns:Publish
  - DynamoDB: PutItem
- DynamoDB 表。

これらのリソースは、AWS コンソールで手動で作成することができますが、このチュートリアルで説明するように AWS CloudFormation 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。

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

AWS CloudFormationは、AWSインフラストラクチャデプロイを予想可能および繰り返し作成し、プロビジョニングすることができます。AWS CloudFormation についてはAWS CloudFormationユーザーガイドを参照してください。

AWS CLI を使用して AWS CloudFormation スタックを作成するには：

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、この GitHub で公開されている AWS CDK を使用して生成されました。AWS CDK の詳細については、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、*`REGION`* を AWS リージョンに置き換えます。

   > **Important**
   >
   > スタック名は、AWS 地域および 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 --region REGION
   ```

   `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 User Guide(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、AWSマネジメントコンソールでAWS CloudFormationを開き、スタックを選択し、Resources（リソース）タブを選択します。
4. スタックが作成されたら、表に入力しますの説明のようにAWS SDK for JavaScriptを使用してDynamoDB表に入力します。

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。*REGION* を実際の AWS リージョンに置き換え、*IDENTITY_POOL_ID* を Amazon Cognito アイデンティティプール ID に置き換えます。これにより DynamoDB クライアントオブジェクトが作成されます。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon DynaomDB service client object.
const dynamoClient = new DynamoDBClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { dynamoClient };
```

このコードはこのGitHubに で利用できます。

次に、プロジェクトフォルダのdynamoAppHelperFilesフォルダを作成、そこにファイルupdate-table.jsを作成し、このGitHubにコンテンツをコピーしてください。

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

// Set the parameters
export const params = {
  TableName: "Items",
  Item: {
    id: { N: "1" },
    title: { S: "aTitle" },
    name: { S: "aName" },
```

```
    body: { S: "aBody" },
  },
};

export const run = async () => {
  try {
    const data = await dynamoClient.send(new PutItemCommand(params));
    console.log("success");
    console.log(data);
  } catch (err) {
    console.error(err);
  }
};
run();
```

コマンドラインから、以下のコマンドを実行します。

```
node update-table.js
```

このコードはこのGitHubにで利用できます。

アプリケーションのフロントエンドページを作成します

ここでは、アプリケーションのフロントエンド HTMLブラウザページを作成します。

DynamoDBApp ディレクトリを作成し、index.html という名前のファイルを作成し、GitHub のここ からコードをコピーします。script要素は、例に必要なすべてのJavaScriptを含むmain.jsファイルを追加します。このチュートリアルの後半で、main.jsファイルを作成します。index.htmlの残りのコードは、ユーザーが入力するデータをキャプチャするブラウザページを作成します。

このサンプルコードは、この GitHub にで見つけられます。

ブラウザスクリプトを作成する

まず、この例に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。libsディレクトリの作成、snsClient.jsを作成し、それに以下のコードをペーストします。それぞれの*REGION*（地域）と*IDENTITY_POOL_ID*（アイデンティティプールID)を置き換えます。

> **Note**
>
> AWS リソースを作成する 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 { SNSClient } from "@aws-sdk/client-sns";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const snsClient = new SNSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { snsClient };
```

このコードは このGitHubに で利用可能です。

この例のブラウザースクリプトを作成するには、DynamoDBAppというフォルダに、ファイル名add_data.jsで Node.js モジュールを作成し、それに以下のコードをペーストします。submitData関数は DynamoDB 表にデータを送信し、Amazon SNSを使用してアプリケーション管理者にSMSテキストを送信します。

submitData関数で、ターゲットの電話番号、アプリケーションインターフェイスで入力された値、および Amazon S3 バケットの名前の変数を表します。次に、表に項目を追加するためのパラメータオブジェクトを作成します。いずれの値も空でない場合は、submitDataが表に項目を追加し、メッセージが送信されます。関数を window.submitData = submitData でブラウザに利用可能にすることを忘れないでください。

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { PublishCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";
import { dynamoClient } from "../libs/dynamoClient.js";

export const submitData = async () => {
  //Set the parameters
  // Capture the values entered in each field in the browser (by id).
  const id = document.getElementById("id").value;
  const title = document.getElementById("title").value;
  const name = document.getElementById("name").value;
  const body = document.getElementById("body").value;
  //Set the table name.
  const tableName = "Items";

  //Set the parameters for the table
  const params = {
    TableName: tableName,
    // Define the attributes and values of the item to be added. Adding ' + "" '
 converts a value to
    // a string.
    Item: {
      id: { N: id + "" },
      title: { S: title + "" },
      name: { S: name + "" },
      body: { S: body + "" },
    },
  };
  // Check that all the fields are completed.
  if (id != "" && title != "" && name != "" && body != "") {
    try {
      //Upload the item to the table
      await dynamoClient.send(new PutItemCommand(params));
      alert("Data added to table.");
      try {
        // Create the message parameters object.
        const messageParams = {
          Message: "A new item with ID value was added to the DynamoDB",
          PhoneNumber: "PHONE_NUMBER", //PHONE_NUMBER, in the E.164 phone number
 structure.
          // For example, ak standard local formatted number, such as (415) 555-2671,
 is +14155552671 in E.164
          // format, where '1' in the country code.
        };
        // Send the SNS message
        const data = await snsClient.send(new PublishCommand(messageParams));
```

```
        console.log(
          "Success, message published. MessageID is " + data.MessageId,
        );
      } catch (err) {
        // Display error message if error is not sent
        console.error(err, err.stack);
      }
    } catch (err) {
      // Display error message if item is no added to table
      console.error(
        "An error occurred. Check the console for further information",
        err,
      );
    }
    // Display alert if all field are not completed.
  } else {
    alert("Enter data in each field.");
  }
};
// Expose the function to the browser
window.submitData = submitData;
```

このサンプルコードは、このGitHubにで見つけられます。

最後に、コマンドプロンプトで以下を実行して、この例の JavaScript をmain.jsという名前のファイルにバンドルします。

```
webpack add_data.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> webpackのインストールについては、「Webpack でアプリケーションをバンドルする」を参照してください。

アプリケーションを実行するには、ブラウザでindex.htmlを開きます。

リソースを削除します

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 スタックを以下のように削除します。

1. AWS マネジメントコンソールで AWS CloudFormation 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 クロスサービスの例の詳細については、「AWS SDK for JavaScript クロスサービスの例」を参照してください。

API Gateway を使用した Lambdaを呼び出し

REST、HTTP、および WebSocket API を大規模に作成、公開、維持、モニタリング、保護するための AWS のサービスである、Amazon API Gateway を使用して、Lambda 関数を呼び出すことができます。API 開発者は、AWS または他のウェブサービス、AWS クラウドに保存されているデータにアクセスする API 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 はサーバーをプロビジョニングしたり管理しなくてもコードを実行でき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。AWS Lambdaの詳細については、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、特定のユースケースを実行する異なる AWS サービスを呼び出します。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。

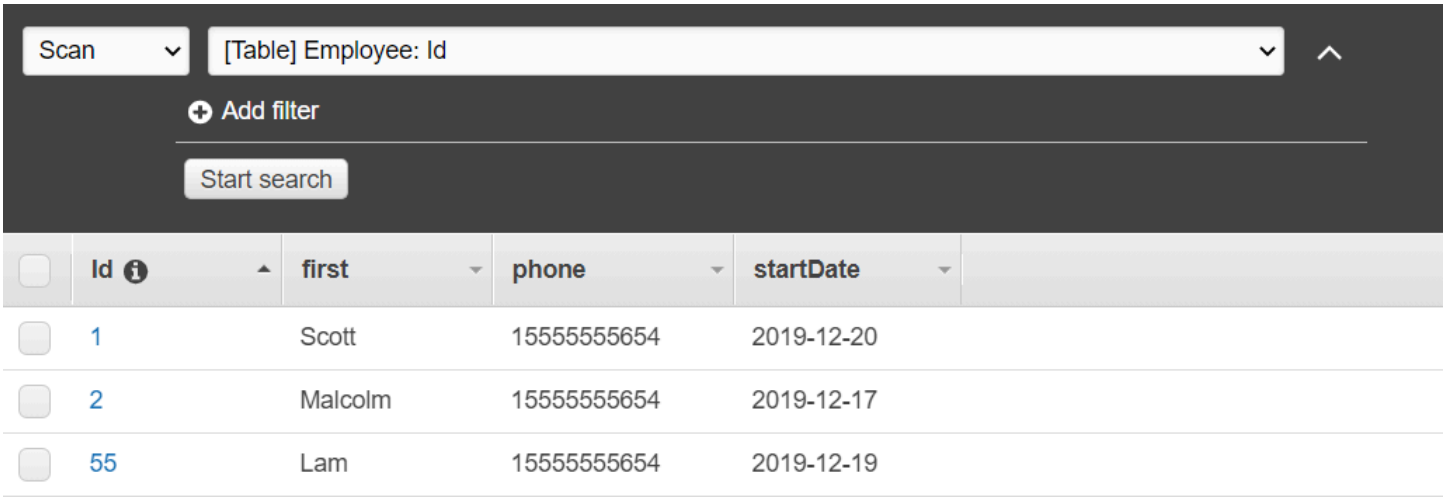
この例では、JavaScript ロジックを使用して、このユースケースを実行するソリューションを作成する方法を示しています。例えば、データベースを読み取り、1 年記念日になった従業員を特定する方

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restエンドポイントに使用し、このAWS Lambda機能を呼び出す方法を説明します。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従業員という名前の Amazon DynamoDB 表を使用します。

- 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**
>
> 完了するためのコスト:このドキュメントに含まれるAWSサービスは、AWSFree Tier (無料利用枠) に含まれます。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リソースを作成します
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設定し、必要な AWS SDK for JavaScript とサードパーティーの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 リソースを作成します

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。

このリソースは手動でも作成できますが、このチュートリアルで説明するように AWS CloudFormation 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。

AWS CloudFormationを使用してAWSリソースを作成します

AWS CloudFormationは、AWSインフラストラクチャデプロイを予想可能および繰り返し作成し、プロビジョニングすることができます。AWS CloudFormation についてはAWS CloudFormationユーザーガイドを参照してください。

AWS CLI を使用して AWS CloudFormation スタックを作成するには：

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、この GitHub で公開されている AWS CDK を使用して生成されました。AWS CDKの詳細については、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 地域および 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地域に置き換えます。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地域に置き換えます。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 リソースを作成します 」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)。

```
"use strict";
const { ScanCommand } = require("@aws-sdk/client-dynamodb");
const { PublishCommand } = require("@aws-sdk/client-sns");
const {snsClient} = require ( "./libs/snsClient" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
```

```
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "Employees",
};
```

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));
    data.Items.forEach(function (element) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
            "Hi " +
```

```
            element.firstName.S +
            "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Lambda 関数をバンドルします

このトピックでは、この例の `mylambdafunction.ts` と必要な AWS SDK for JavaScript のモジュールを `index.js` いうバンドルファイルにバンドルする方法について説明します。

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、「Webpack でアプリケーションをバンドルする」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 リソースを作成します 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 リソースを作成します 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

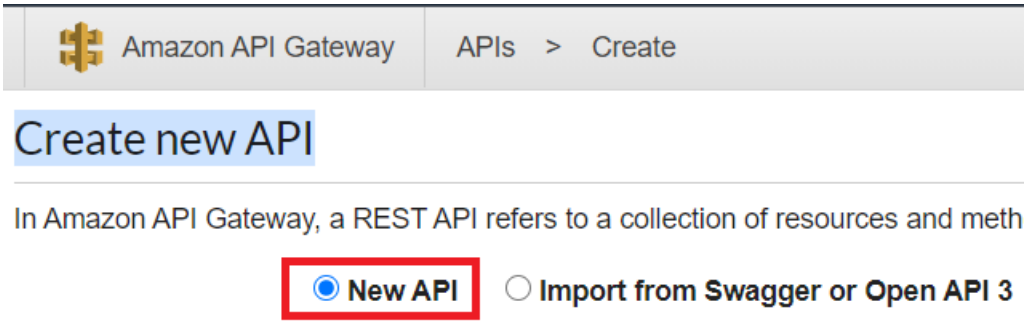
アプリを構築するには

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

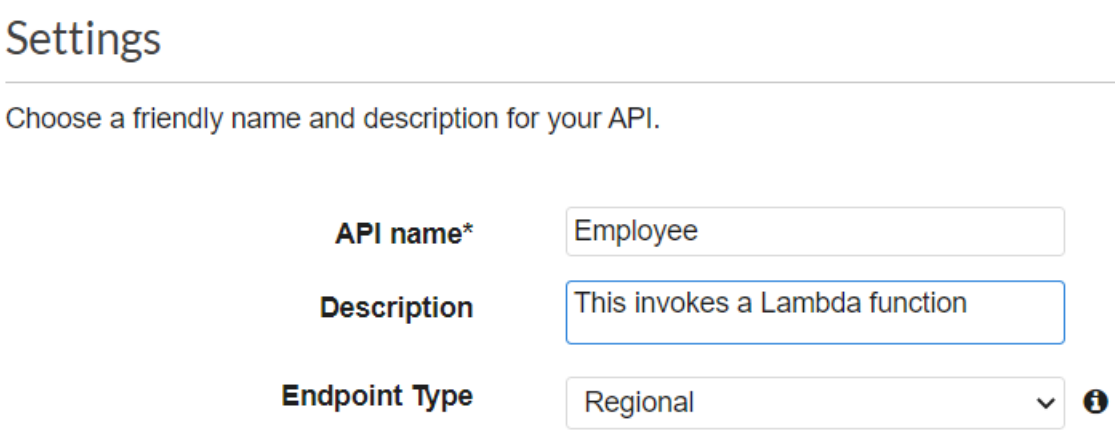
rest API を作成する

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。

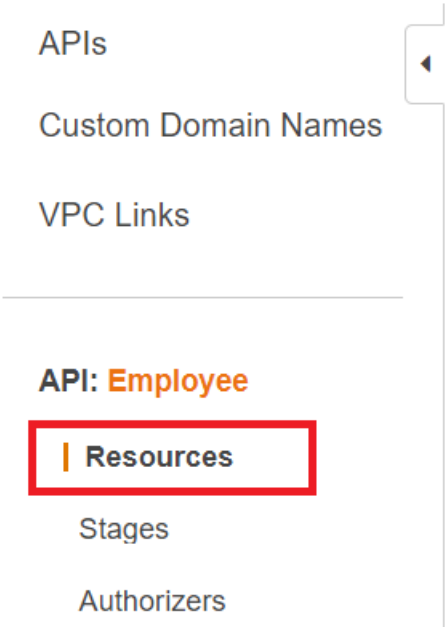
1. [Amazon API Gateway console](Amazon API Gateway コンソール)にサインインします。
2. REST APIで、 [Build] (　構築　) を選択します。
3. [New API] (　新規API)を選択します。



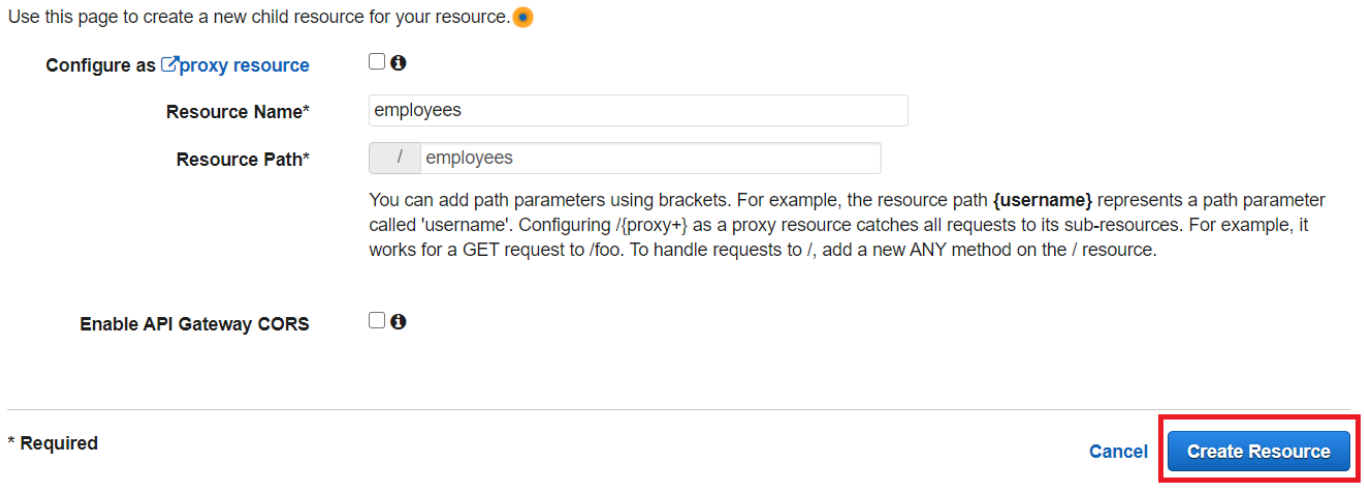
4. [Employee]をAPI 名として指定し、説明を入力します。



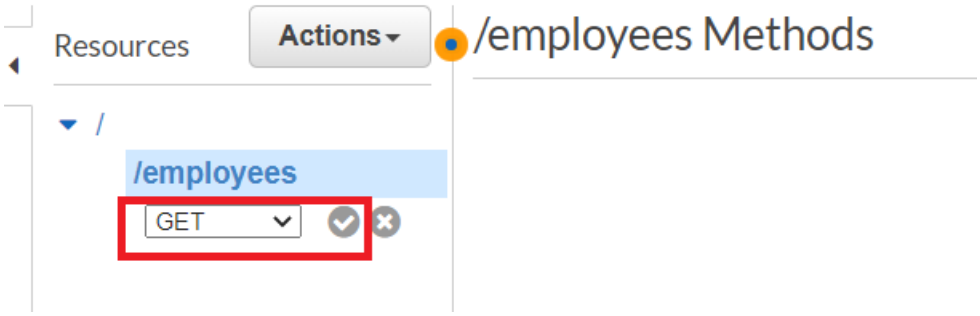
5. API の作成 を選択します。
6. Employeeセクションの[Resources]を選択します。



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。



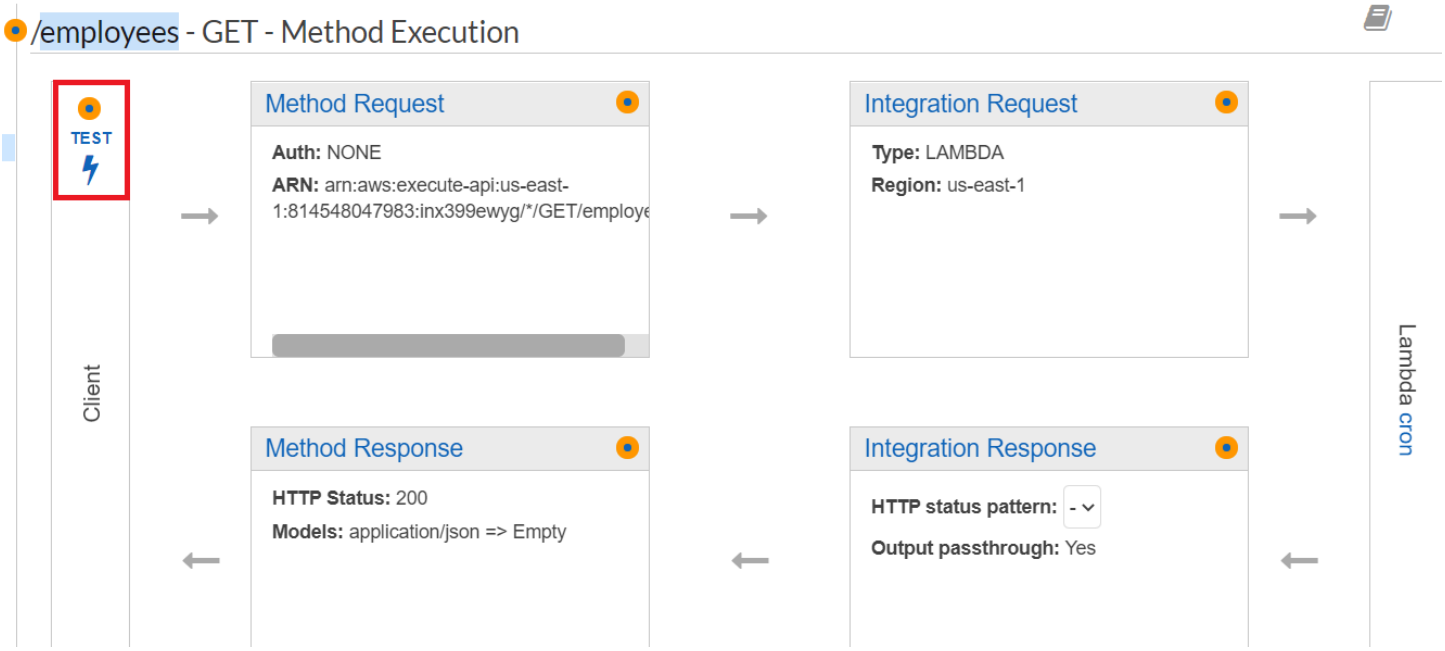
10. [/employees]を選択し、[Create Method] から[Actions]を選択し、[GET]を[/employees]下のドロップダウンメニューから選択します。チェックマークアイコンを選択します。



11. Lambda functionを選択し、Lambda 関数名としてmylambdafunctionと入力します。[Save (保存)] 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。

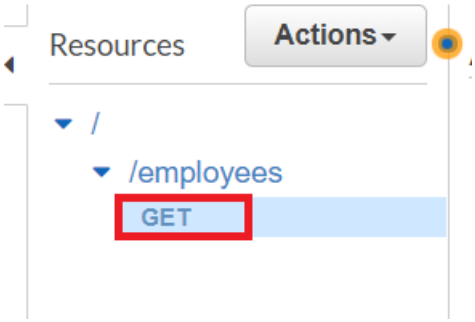


Lambda 関数が呼び出されると、ログファイルを表示して成功したメッセージを表示できます。

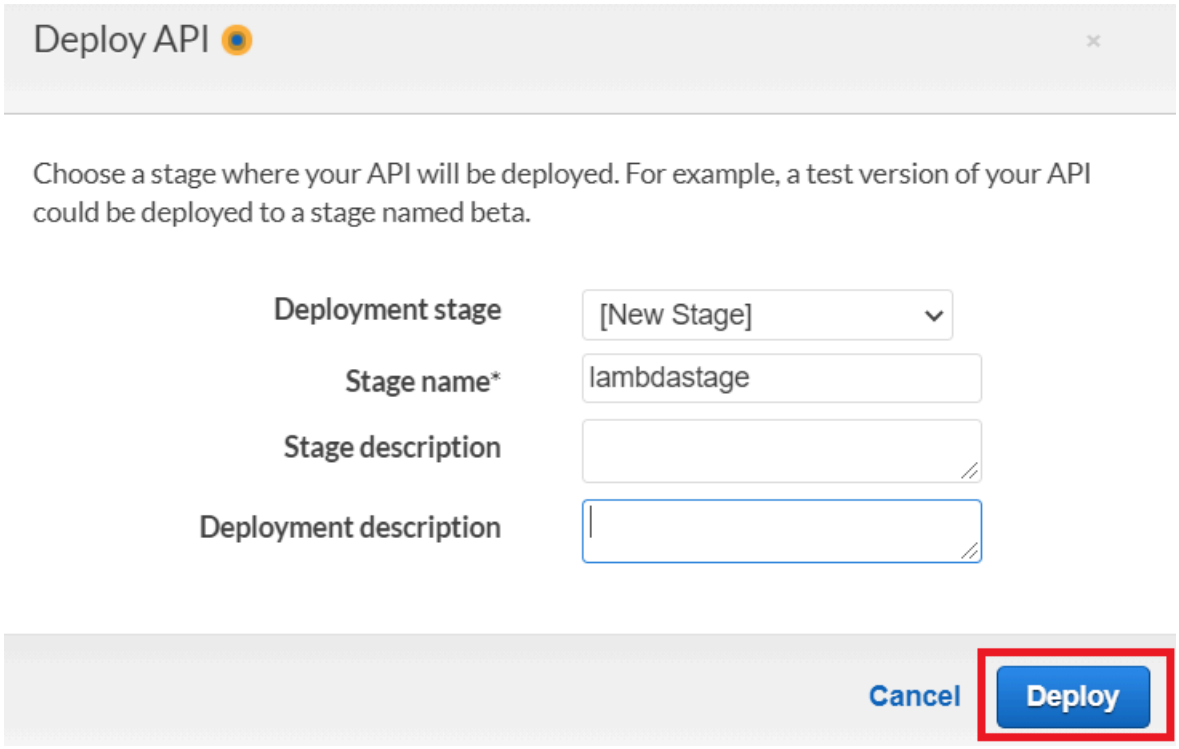
API Gateway メソッドをデプロイする

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。

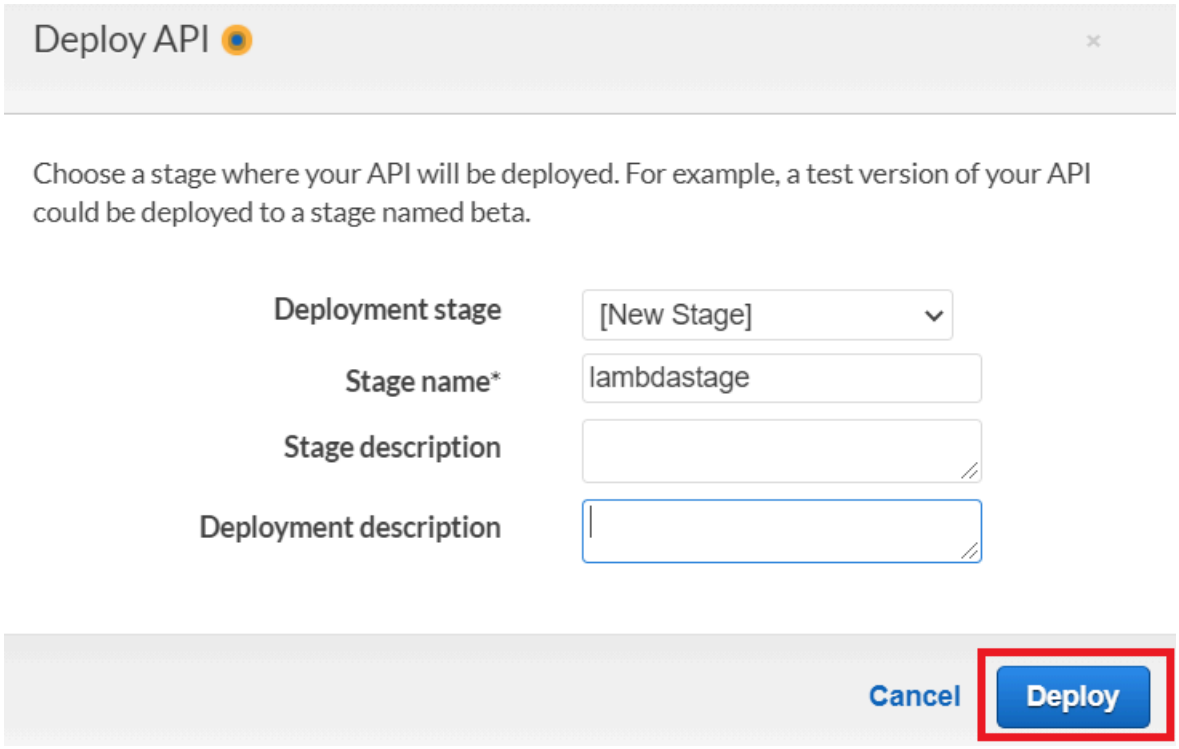
1. [GET] (取得する) を選択します。



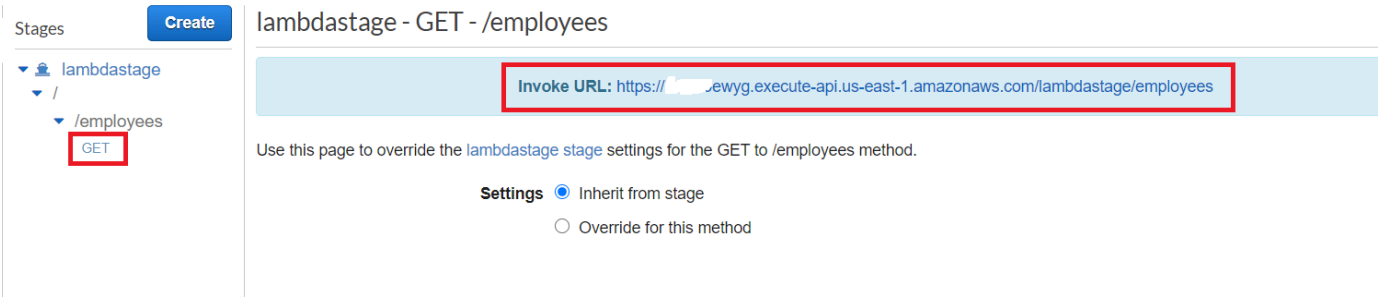
2. [Actions] (アクション) ドロップダウンから[Deploy API](デプロイAPI) を選択します。



3. [Deploy API]フォームに入力し、[Deploy]を選択します。



4. [Save changes] (変更の保存) をクリックします。
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 リソースを作成します トピックで作成した AWS CloudFormation スタックを以下のように削除します。

1. AWS マネジメントコンソールで AWS CloudFormation 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. [Delete] (削除) をクリックします。

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を作成する

Amazon 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 CloudWatch 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、Lambda 関数を毎日呼び出すように CloudWatch イベントをスケジュールできます。

AWS Lambda はサーバーをプロビジョニングしたり管理しなくてもコードを実行でき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。AWS Lambdaの詳細については、とはAWS Lambdaを参照してください。

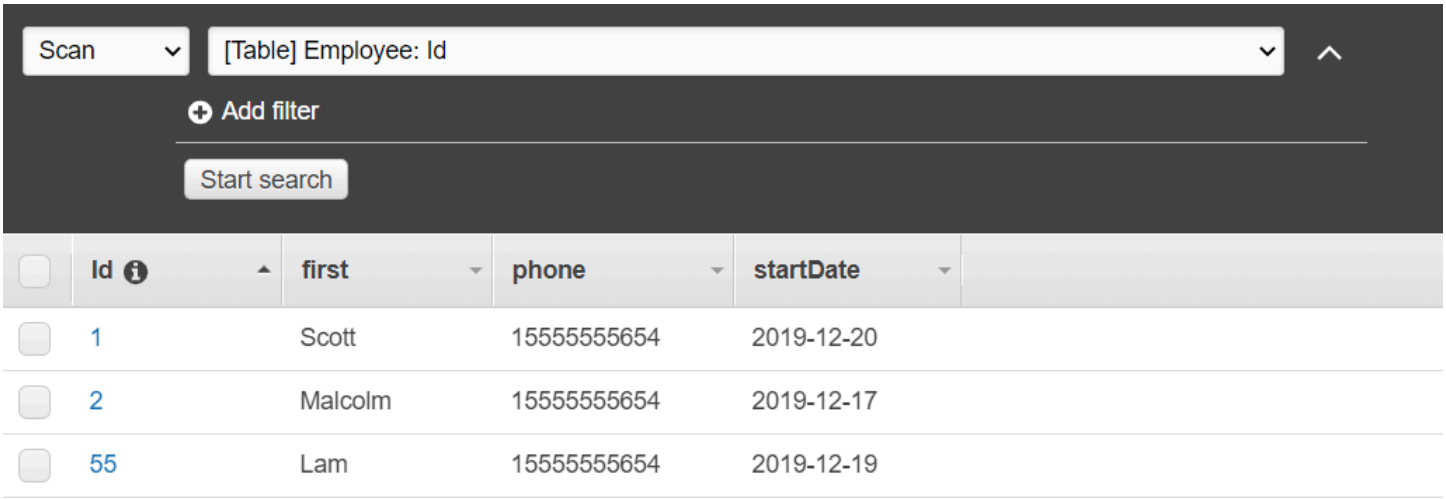
このチュートリアルでは、Lambda ランタイム API を使用して Lambda JavaScript 関数を作成します。この例では、特定のユースケースを実行する異なる AWS サービスを呼び出します。例えば、次の図に示すように、組織が 1 周年記念日に従業員を祝福するモバイルテキストメッセージを送信するとします。

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従業員という Amazon DynamoDB 表を使用します。

- 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**
>
> 完了するためのコスト、このドキュメントに含まれる AWSサービスは、 AWSFree Tier (無料利用枠) に含まれます。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リソースを作成します
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。「」の手順に従いますGitHub。
- 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 リソースを作成します

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。

このリソースは手動でも作成できますが、このチュートリアルで説明するように AWS CloudFormation 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。

AWS CloudFormationを使用してAWSリソースを作成します

AWS CloudFormationは、AWSインフラストラクチャデプロイを予想可能および繰り返し作成し、プロビジョニングすることができます。AWS CloudFormation についてはAWS CloudFormationユーザーガイドを参照してください。

AWS CLI を使用して AWS CloudFormation スタックを作成するには：

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setup.yamlに という名前のファイルを作成し、そこに GitHubコンテンツをコピーします。

> **Note**
>
> AWS CloudFormation テンプレートは、 でAWS CDK入手可能な を使用して生成されました。 GitHubAWS CDKの詳細については、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 地域および 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 User Guide」 (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、Resources (リソース) タブを選択して、コンソールに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、DynamoDB 表にデータを入力します。で説明されているように、AWS SDK for JavaScriptを使用して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、 にあります GitHub。

次に、プロジェクトフォルダのルートディレクトリpopulate-table.jsに という名前のファイルを作成し、そこにコンテンツをコピー GitHub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、 にあります GitHub。

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、DynamoDBClientとDynamoDBScanCommandおよびSNSClientとAmazon SNS PublishCommandコマンドをインポートします。*REGION*をAWS地域に置き換えます。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 リソースを作成します 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、この例の mylambdafunction.js と必要な AWS SDK for JavaScript のモジュールを index.js いうバンドルファイルにバンドルする方法について説明します。

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、Webpack でアプリケーションをバンドルする 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 JavaScript の 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 リソースを作成します 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 リソースを作成します 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、 で GitHub入手できます。

Lambda 関数を呼び出す CloudWatch ように を設定する

Lambda 関数を呼び出す CloudWatch ように 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. トリガータイプを 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、「 CloudWatch イベントで Lambda を使用する」を参照してください。

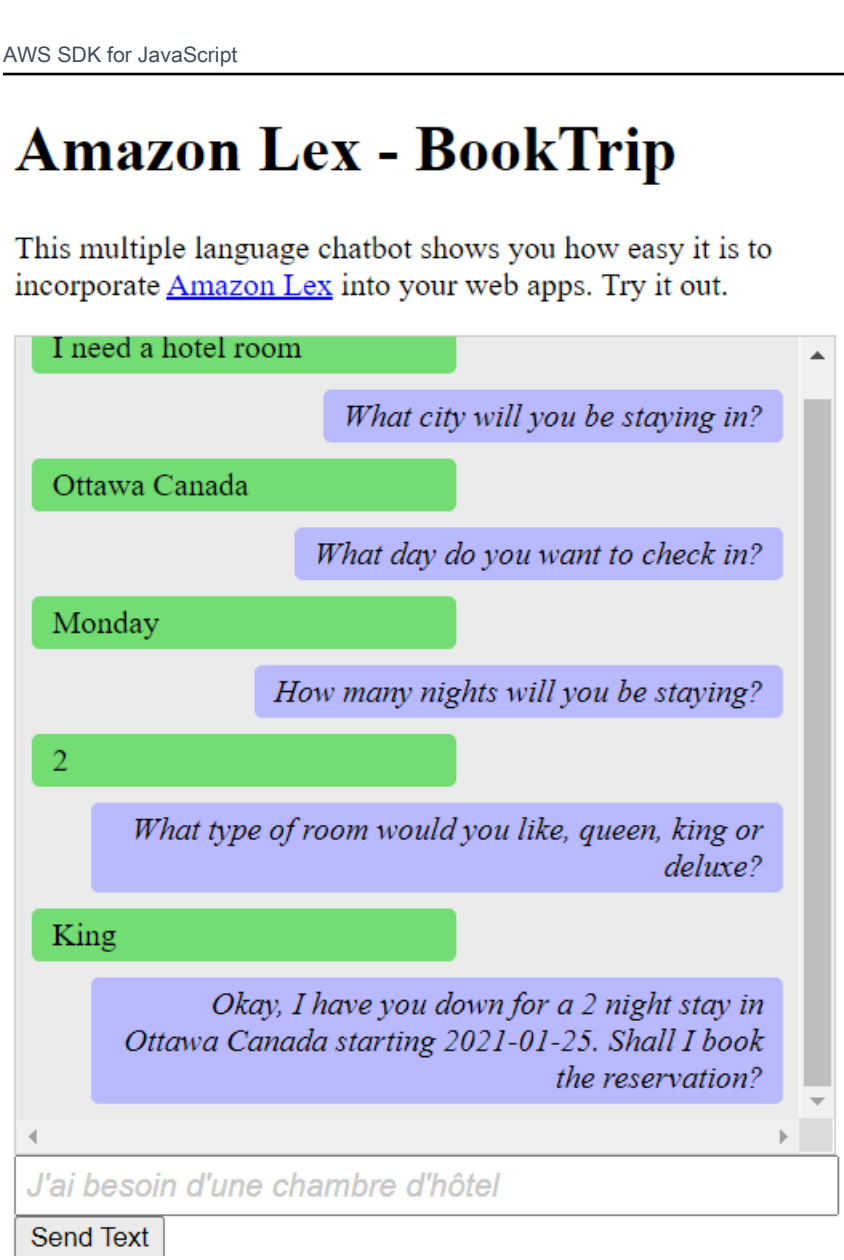
リソースを削除します

お疲れ様でした。を使用して、Amazon CloudWatch のスケジュールされたイベントを通じて Lambda 関数を呼び出しましたAWS SDK for JavaScript。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 リソースを作成します トピックで作成した AWS CloudFormation スタックを以下のように削除します。

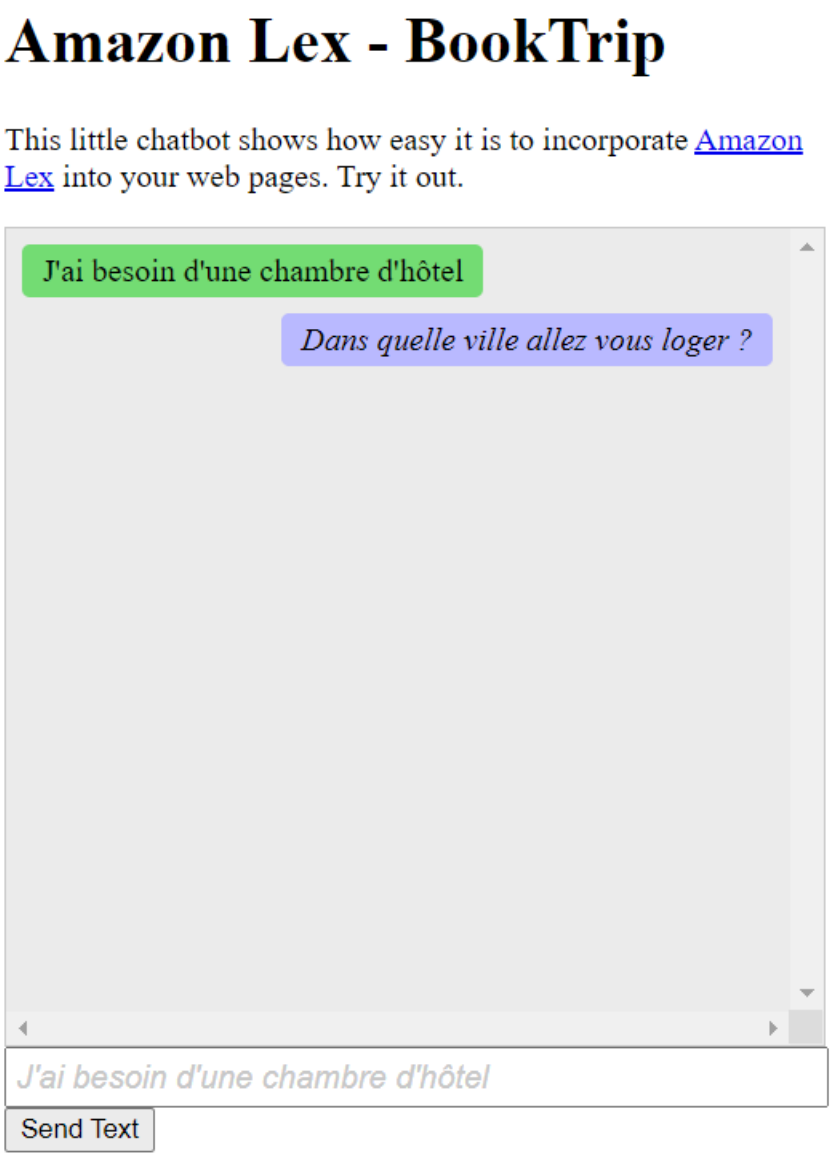
1. AWS CloudFormation コンソールを開きます。
2. 「スタック」ページで、スタックを選択します。
3. [削除] をクリックします。

Amazon Lex chatbotを構築する

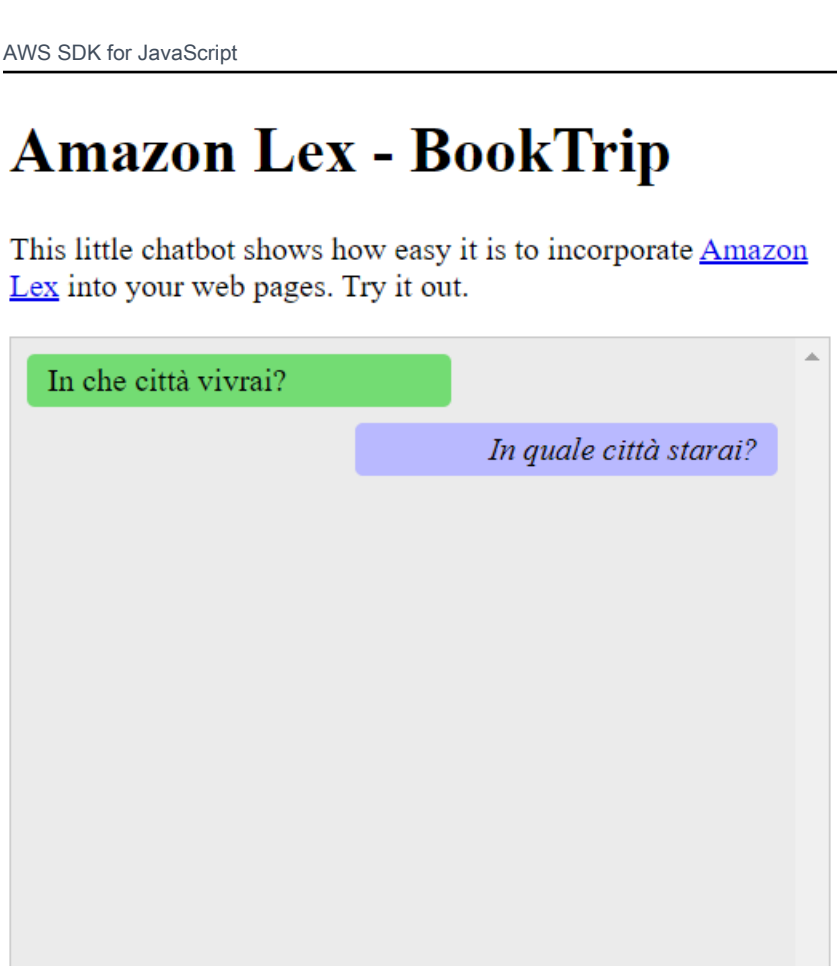
ウェブアプリケーション内に Amazon Lex chatbotを作成して、ウェブサイトの訪問者に対応することができます。Amazon Lex chatbotは、人と直接連絡することなく、ユーザーとのオンラインチャットの会話を実行する機能です。例えば、次の図は、ホテルの部屋の予約についてユーザーに対応する Amazon Lex chatbotを示しています。



このAWSのチュートリアルで作成されたAmazon Lex chatbot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 chatbot を作成し、それを 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、次の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

完了するためのコスト。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。「」の手順に従いますGitHub。
- 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 リソースを作成します

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も作成できますが、このチュートリアルで説明するようにAWS CloudFormation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。

AWS CloudFormationを使用してAWSリソースを作成する

AWS CloudFormationは、AWSインフラストラクチャデプロイを予想可能および繰り返し作成し、プロビジョニングすることができます。AWS CloudFormation についてはAWS CloudFormationユーザーガイドを参照してください。

AWS CLI を使用して AWS CloudFormation スタックを作成するには：

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`setup.yaml`に という名前のファイルを作成し、そこに GitHubコンテンツをコピーします。

> **Note**
>
> AWS CloudFormation テンプレートは、 でAWS CDK入手可能な を使用して生成されました。 GitHubAWS CDKの詳細については、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 地域および 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。

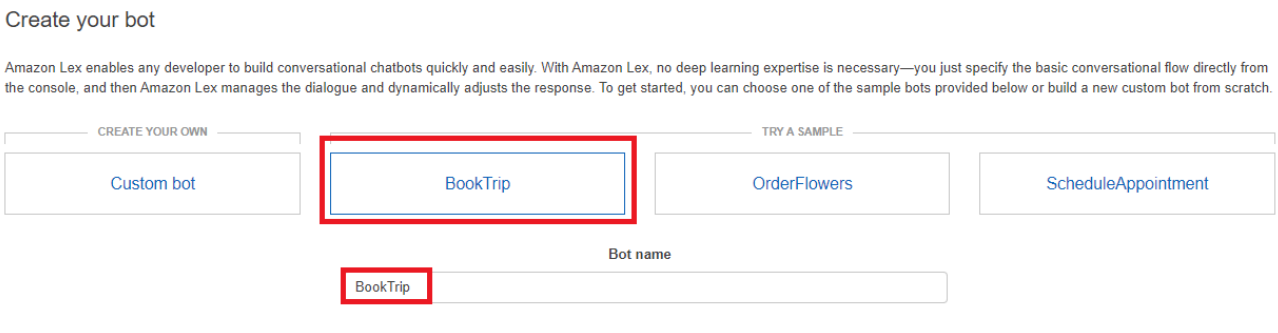
作成されたリソースを表示するには、Amazon Lex コンソールを開き、スタックを選択し、Resources (リソース)タブを選びます。

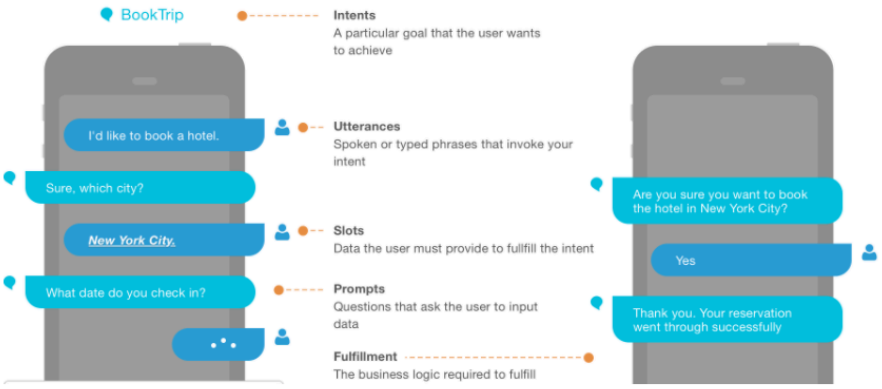
Amazon Lex botを作成します

> **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。

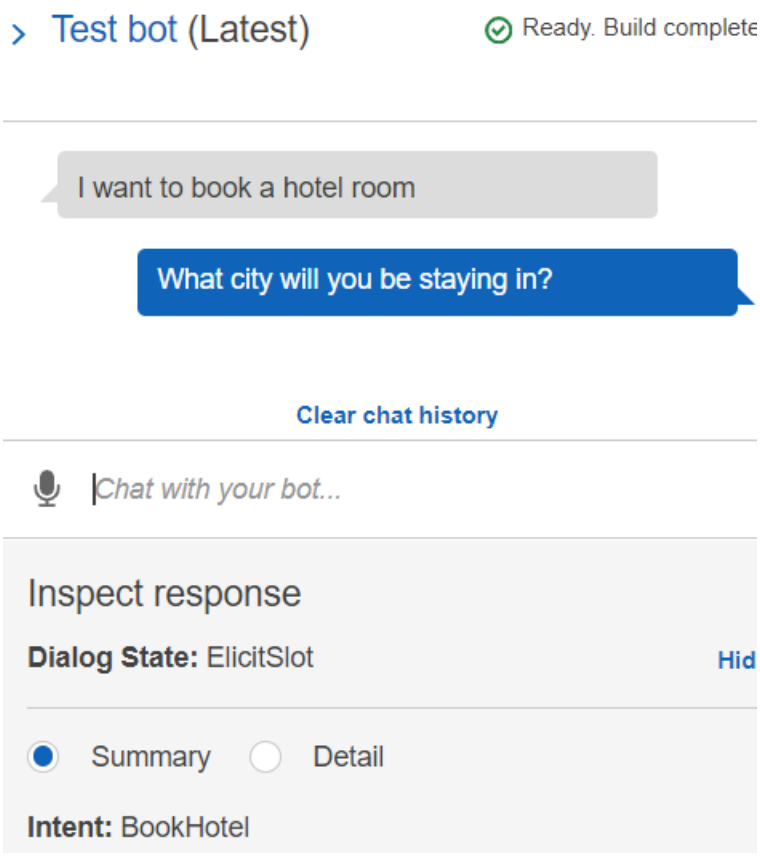
最初のステップは、Amazon Web Services マネジメントコンソールを使用して Amazon Lex chatbot を作成することです。この例では、Amazon LexBookTrip の例を使用しています。詳細については「Book Trip (出張の予約)」を参照してください。

- 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 設計図を選択します (デフォルトのボット名は のままにしますBookTrip) 。





- デフォルト設定を入力し、作成 (コンソールにボットが表示されます) BookTrip 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。



- Publish (発行) を選択し、エイリアス名を識別します (AWS SDK for JavaScriptを使用する場合、この値が必要です)。

> **Note**
>
> JavaScript コード内でボット名と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>
```

このコードは、 でも GitHub入手できます。

ブラウザスクリプトを作成

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リージョンに置き換え、*IDENTITY_POOL_ID*をAWS リソースを作成します で作成したアイデンティティプールのID に置き換えます。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**
>
> AWS CloudFormationを使用してAWSリソースを作成するで作成した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、 にあります GitHub。

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";

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

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

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

  var lexTextResponse = lexResponse;

  responsePara.appendChild(document.createTextNode(lexTextResponse));
  responsePara.appendChild(document.createElement("br"));
  conversationDiv.appendChild(responsePara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function handletext(text) {
  g_text = text;
  var xhr = new XMLHttpRequest();
  xhr.addEventListener("load", loadNewItems, false);
  xhr.open("POST", "../text", true); // A Spring MVC controller
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded"); //
necessary
  xhr.send("text=" + text);
}
```

```
function loadNewItems() {
  showRequest();

  // Re-enable input.
  var wisdomText = document.getElementById("wisdom");
  wisdomText.value = "";
  wisdomText.locked = false;
}

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

    const comprehendParams = {
      Text: wisdom,
    };
    try {
      const data = await comprehendClient.send(
        new DetectDominantLanguageCommand(comprehendParams)
      );
      console.log(
        "Success. The language code is: ",
        data.Languages[0].LanguageCode
      );
      const translateParams = {
        SourceLanguageCode: data.Languages[0].LanguageCode,
        TargetLanguageCode: "en", // For example, "en" for English.
        Text: wisdom,
      };
      try {
        const data = await translateClient.send(
          new TranslateTextCommand(translateParams)
        );
        console.log("Success. Translated text: ", data.TranslatedText);
        const lexParams = {
          botName: "BookTrip",
          botAlias: "mynewalias",
```

```
          inputText: data.TranslatedText,
          userId: "chatbot", // For example, 'chatbot-demo'.
        };
        try {
          const data = await lexClient.send(new PostTextCommand(lexParams));
          console.log("Success. Response is: ", data.message);
          var msg = data.message;
          showResponse(msg);
        } catch (err) {
          console.log("Error responding to message. ", err);
        }
      } catch (err) {
        console.log("Error translating text. ", err);
      }
    } catch (err) {
      console.log("Error identifying language. ", err);
    }
  }
};
// Make the function available to the browser.
window.createResponse = createResponse;
```

このコードは、 にあります GitHub。

ここで、webpackを使用して、`index.js`とAWS SDK for JavaScriptのモジュールを 1つのファイル`main.js`にバンドルします。

1. webpackのインストールが済んでいない場合は、この例の前提条件に従ってインストールしてください。

   > **Note**
   >
   > Webpack の詳細については、Webpack でアプリケーションをバンドルする 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 JavaScript の を というファイルにバンドルします`main.js`。

   ```
   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 リソースを作成します トピックで作成した AWS CloudFormation スタックを以下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の詳細については、「AWS SDK for JavaScript クロスサービスの例」を参照してください。

メッセージングアプリケーション例の作成

AWS SDK for JavaScriptおよびAmazon Simple Queue Service (Amazon SQS) を使用して、メッセージの送信や取得を行うAWSアプリケーションを作成することもできます。メッセージは、メッセージの順序が一貫していることを保証する先入れ先出し (FIFO) キューに保存されます。例えば、キューに保存される最初のメッセージは、キューから読み込まれた最初のメッセージです。

> **Note**
>
> Amazon SQS の詳細については、「What is Amazon Simple Queue Service?」（Amazon Simple Queue Serviceとは？）を参照してください。

このチュートリアルでは、AWSメッセージングという名前のNode.jsアプリケーションを作成します。

完了するためのコスト。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. ワークフローを理解する
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。「」の手順に従いますGitHub。
- 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 リソースを作成します

このチュートリアルでは、以下のリソースが必要です。

- Amazon SQS のアクセス権限を持つ非認証 IAM ロール。
- Message.FIFO (メッセージFIFO) という名前の FIFO Amazon SQS キュー - キューの作成についてはAmazon SQS キューの作成を参照してください。

このリソースは手動で作成することもできますが、このチュートリアルで説明するようにAWS CloudFormation(AWS CloudFormation)を使用して、これらのリーソースをプロビジョニングすることをお勧めします。

> **Note**
>
> AWS CloudFormationはソフトウェア開発フレームワークで、クラウドアプリケーションのリソースを定義します。詳細については、『AWS CloudFormation ユーザーガイド』を参照してください。

AWS CloudFormation を使用して AWS リソースを作成します

AWS CloudFormation は、AWS インフラストラクチャデプロイを予想可能および繰り返し作成し、プロビジョニングすることができます。AWS CloudFormation についてはAWS CloudFormationユーザーガイドを参照してください。

AWS CLI を使用して AWS CloudFormation スタックを作成するには：

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`setup.yaml`に という名前のファイルを作成し、そこに GitHubコンテンツをコピーします。

   > **Note**
   >
   > AWS CloudFormation テンプレートは、 でAWS CDK入手可能な を使用して生成されました。 GitHubAWS CDKの詳細については、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 地域および 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 User Guide(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、AWSマネジメントコンソールでAWS CloudFormationを開き、スタックを選択し、Resources(リソース) タブを選択します。

AWSメッセージングアプリケーションを理解する

SQS キューにメッセージを送信するには、アプリケーションにメッセージを入力し、送信を選択します。

メッセージが送信されると、アプリケーションはメッセージを表示します。

PURGE (パージ) を選択するとAmazon SQS キューからのメッセージをパージすることができます。これにより、キューが空になり、アプリケーションにはメッセージが表示されません。

次に、アプリケーションがメッセージをどのように処理するかを説明します。

- ユーザは自分の名前を選択してメッセージを入力し、メッセージを送信します。これにより、`pushMessage`関数が開始します。
- `pushMessage`は、Amazon SQS キュー Urlを取得し、一意のメッセージ ID 値 (GUID)、メッセージテキスト、およびユーザーを含むメッセージをAmazon SQS キューに送信します。
- `pushMessage`は、Amazon SQS キューからメッセージを取得し、各メッセージのユーザーとメッセージを抽出し、メッセージを表示します。
- ユーザーはメッセージをパージできます。これにより、Amazon SQS キューとユーザーインターフェイスからメッセージを削除します。

HTMLページを作成する

次に、アプリケーションのグラフィカルユーザーインターフェイス (GUI) に必要な HTML ファイルを作成します。`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ます。

```
<!doctype html>
<html
  xmlns:th="http://www.thymeleaf.org"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" href="./images/favicon.ico" />
    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    />
    <link rel="stylesheet" href="./css/styles.css" />
    <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
    <script src="https://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
    <script src="./js/main.js"></script>
    <style>
      .messageelement {
        margin: auto;
        border: 2px solid #dedede;
        background-color: #d7d1d0;
        border-radius: 5px;
        max-width: 800px;
        padding: 10px;
        margin: 10px 0;
      }

      .messageelement::after {
        content: "";
        clear: both;
        display: table;
      }

      .messageelement img {
        float: left;
        max-width: 60px;
        width: 100%;
        margin-right: 20px;
```

```
        border-radius: 50%;
      }

      .messageelement img.right {
        float: right;
        margin-left: 20px;
        margin-right: 0;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h2>AWS Sample Messaging Application</h2>
      <div id="messages"></div>

      <div class="input-group mb-3">
        <div class="input-group-prepend">
          <span class="input-group-text" id="basic-addon1">Sender:</span>
        </div>
        <select name="cars" id="username">
          <option value="Scott">Brian</option>
          <option value="Tricia">Tricia</option>
        </select>
      </div>

      <div class="input-group">
        <div class="input-group-prepend">
          <span class="input-group-text">Message:</span>
        </div>
        <textarea
          class="form-control"
          id="textarea"
          aria-label="With textarea"
        ></textarea>
        <button
          type="button"
          onclick="pushMessage()"
          id="send"
          class="btn btn-success"
        >
          Send
        </button>
        <button
          type="button"
```

```
          onclick="purge()"
          id="refresh"
          class="btn btn-success"
        >
          Purge
        </button>
      </div>
      <!-- All of these child items are hidden and only displayed in a FancyBox
 ------------------------------------------------------>
      <div id="hide" style="display: none">
        <div id="base" class="messageelement">
          <img
            src="../public/images/av2.png"
            alt="Avatar"
            class="right"
            style="width: 100%"
          />
          <p id="text">Excellent! So, what do you want to do today?</p>
          <span class="time-right">11:02</span>
        </div>
      </div>
    </div>
  </body>
</html>
```

このコードは、 でも GitHub入手できます。

ブラウザスクリプトの作成

このトピックでは、アプリケーションのブラウザスクリプトを作成します。ブラウザスクリプトを作成したら、それをのバンドル JavaScriptで説明したようにmain.jsというファイルにバンドルします。

index.js という名前のファイルを作成します。ここからコードをコピーして貼り付け GitHubます。

このコードについては、次のセクションで説明します。

1. 設定
2. populateChat
3. pushmessages
4. PURGE (パージ)

構成

まず、libsディレクトリを作成して、sqsClient.jsという名前のファイルを作成することで、必要な Amazon SQS クライアントオブジェクトを作成します。それぞれの*REGION*と*IDENTITY_POOL_ID*を置き換えます。

> **Note**
>
> AWS リソースを作成します 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-providers";
import {SQSClient} from "@aws-sdk/client-sqs";
const REGION = "REGION"; //e.g. "us-east-1"
const IdentityPoolId = "IDENTITY_POOL_ID";
const sqsClient = new SQSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IdentityPoolId
  }),
});
```

index.jsで、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。*SQS_QUEUE_NAME*をAWS リソースを作成します で作成した Amazon SQS キューの名前に置き換えます。

```
import {
  GetQueueUrlCommand,
  SendMessageCommand,
  ReceiveMessageCommand,
  PurgeQueueCommand,
} from "@aws-sdk/client-sqs";
import { sqsClient } from "./libs/sqsClient.js";

const QueueName = "SQS_QUEUE_NAME"; // The Amazon SQS queue name, which must end
 in .fifo for this example.
```

populateChat

populateChat関数 onload は、Amazon SQS キューの URL を自動的に取得し、キューのすべてのメッセージを取得して表示します。

```
$(function () {
  populateChat();
});

const populateChat = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for retrieving the messages in the Amazon SQS Queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };
    try {
      // Retrieve the messages from the Amazon SQS Queue.
      const data = await sqsClient.send(
        new ReceiveMessageCommand(getMessageParams)
      );
      console.log("Successfully retrieved messages", data.Messages);

      // Loop through messages for user and message body.
      var i;
      for (i = 0; i < data.Messages.length; i++) {
        const name = data.Messages[i].MessageAttributes.Name.StringValue;
        const body = data.Messages[i].Body;
        // Create the HTML for the message.
        var userText = body + "<br><br><b>" + name;
```

```
        var myTextNode = $("#base").clone();
        myTextNode.text(userText);
        var image_url;
        var n = name.localeCompare("Scott");
        if (n == 0) image_url = "./images/av1.png";
        else image_url = "./images/av2.png";
        var images_div =
          '<img src="' +
          image_url +
          '" alt="Avatar" class="right" style=""width:100%;"">';
        myTextNode.html(userText);
        myTextNode.append(images_div);

        // Add the message to the GUI.
        $("#messages").append(myTextNode);
      }
    } catch (err) {
      console.log("Error loading messages: ", err);
    }
  } catch (err) {
    console.log("Error retrieving SQS queue URL: ", err);
  }
};
```

メッセージを転送する

ユーザは自分の名前を選択してメッセージを入力し、メッセージを送信します。これにより、pushMessage関数を開始します。pushMessageは Amazon SQS キュー URLを取得し、一意のメッセージ ID 値 (GUID)とメッセージテキスト、およびユーザーを含むメッセージをAmazon SQS キューに送信します。次に、Amazon SQS キューからすべてのメッセージを取得し、それらを表示します。

```
const pushMessage = async () => {
  // Get and convert user and message input.
  var user = $("#username").val();
  var message = $("#textarea").val();

  // Create random deduplication ID.
  var dt = new Date().getTime();
  var uuid = "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (
    c
  ) {
    var r = (dt + Math.random() * 16) % 16 | 0;
```

```
    dt = Math.floor(dt / 16);
    return (c == "x" ? r : (r & 0x3) | 0x8).toString(16);
  });

  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for the message.
    var messageParams = {
      MessageAttributes: {
        Name: {
          DataType: "String",
          StringValue: user,
        },
      },
      MessageBody: message,
      MessageDeduplicationId: uuid,
      MessageGroupId: "GroupA",
      QueueUrl: data.QueueUrl,
    };
    const result = await sqsClient.send(new SendMessageCommand(messageParams));
    console.log("Success", result.MessageId);

    // Set the parameters for retrieving all messages in the SQS queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };

    // Retrieve messages from SQS Queue.
    const final = await sqsClient.send(
      new ReceiveMessageCommand(getMessageParams)
    );
```

```
    console.log("Successfully retrieved", final.Messages);
    $("#messages").empty();
    // Loop through messages for user and message body.
    var i;
    for (i = 0; i < final.Messages.length; i++) {
      const name = final.Messages[i].MessageAttributes.Name.StringValue;
      const body = final.Messages[i].Body;
      // Create the HTML for the message.
      var userText = body + "<br><br><b>" + name;
      var myTextNode = $("#base").clone();
      myTextNode.text(userText);
      var image_url;
      var n = name.localeCompare("Scott");
      if (n == 0) image_url = "./images/av1.png";
      else image_url = "./images/av2.png";
      var images_div =
        '<img src="' +
        image_url +
        '" alt="Avatar" class="right" style=""width:100%;"">';
      myTextNode.html(userText);
      myTextNode.append(images_div);
      // Add the HTML to the GUI.
      $("#messages").append(myTextNode);
    }
  } catch (err) {
    console.log("Error", err);
  }
};
// Make the function available to the browser window.
window.pushMessage = pushMessage;
```

メッセージをパージする

purgeは、Amazon SQS キューおよびユーザーインターフェイスからメッセージを削除します。

```
// Delete the message from the Amazon SQS queue.
const purge = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
```

```
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success", data.QueueUrl);
    // Delete all the messages in the Amazon SQS Queue.
    const result = await sqsClient.send(
      new PurgeQueueCommand({ QueueUrl: data.QueueUrl })
    );
    // Delete all the messages from the GUI.
    $("#messages").empty();
    console.log("Success. All messages deleted.", data);
  } catch (err) {
    console.log("Error", err);
  }
};

// Make the function available to the browser window.
window.purge = purge;
```

のバンドル JavaScript

この完全なブラウザスクリプトコードは、 にあります GitHub。

ここで、webpackを使用して、`index.js`とAWS SDK for JavaScriptのモジュールを 1つのファイルmain.jsにバンドルします。

1. webpackのインストールが済んでいない場合は、この例の前提条件に従ってインストールしてください。

   > **Note**
   >
   > Webpack の詳細については、Webpack でアプリケーションをバンドルする 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 JavaScript の を <index.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 SQS を使用するAWSメッセージングアプリケーションが作成されデプロイされました。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、それらに対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 リソースを作成します トピックで作成した AWS CloudFormation スタックを以下のように削除します。

1. AWS マネジメントコンソールで AWS CloudFormation 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] をクリックします。

AWS Cloud9 で を使用する AWS SDK for JavaScript

AWS Cloud9 で を使用すると AWS SDK for JavaScript 、ブラウザコード JavaScript で を記述および実行したり、ブラウザだけで Node.js コードを記述、実行、デバッグしたりできます。 AWS Cloud9 には、コードエディタやターミナルなどのツールに加えて、Node.js コードのデバッガーが含まれています。

AWS Cloud9 IDE はクラウドベースのため、オフィス、自宅、またはインターネットに接続されたマシンを使用して、どこからでもプロジェクトを操作できます。の一般的な情報については AWS Cloud9、AWS Cloud9 「 ユーザーガイド」を参照してください。

次のステップでは、 SDK for AWS Cloud9 を使用して を設定する方法について説明します JavaScript。

目次

ステップ 1: を使用するように AWS アカウントを設定する AWS Cloud9

の使用を開始するには、 アカウントの のアクセス許可を持つ AWS Identity and Access Management (IAM) エンティティ (IAM ユーザーなど) AWS Cloud9 としてコンソール AWS Cloud9 に AWS Cloud9 サインインします AWS 。

AWS アカウントで IAM エンティティを設定して にアクセスし AWS Cloud9、 AWS Cloud9 コンソールにサインインするには、 AWS Cloud9 ユーザーガイドの「 のチーム設定 AWS Cloud9」を参照してください。

ステップ 2: AWS Cloud9 開発環境を設定する

AWS Cloud9 コンソールにサインインしたら、 コンソールを使用して AWS Cloud9 開発環境を作成します。環境を作成すると、 はその環境の IDE AWS Cloud9 を開きます。

詳細については、AWS Cloud9 User Guide(ユーザーガイド)のCreating an environment in AWS Cloud9(環境の作成)を参照してください。

> **Note**
>
> コンソールで始めて環境を作成する際に、[Create a new instance for environment (EC2)] (環境の新しいインスタンスを作成する) (EC2) オプションを選択することをお勧めします。このオプションは、 AWS Cloud9 に環境を作成し、Amazon EC2 インスタンスを起動して、新しいインスタンスを新しい環境に接続するように指示します。これは、 の使用を開始する最も速い方法です AWS Cloud9。

ステップ 3: の SDK をセットアップする JavaScript

が開発環境の IDE AWS Cloud9 を開いたら、次のいずれかまたは両方の手順に従って、IDE を使用して環境で の SDK JavaScript を設定します。

for Node.js JavaScript をセットアップするには

1. IDE でターミナルが開いていない場合は開きます。これを行うには、IDE のメニューバーで、[Window] (ウィンドウ)、[New Terminal] (新しいターミナル) の順に選択します。
2. 次のコマンドを実行して、 npmを使用して SDK for のCloud9クライアントをインストールします JavaScript。

   ```
   npm install @aws-sdk/client-cloud9
   ```

   IDE がnpmを見つけられない場合は、次のコマンドを一度に1つずつ順番に実行して、npmをインストールします。(これらのコマンドでは、このトピックで前述した [Create a new instance for environment (環境の新しいインスタンスを作成する) (EC2)] オプションを選択していることを前提としています)。

> **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 (Node Version Manager) GitHub リポジトリを参照してください。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash #
 Download and install Node Version Manager (nvm).
. ~/.bashrc                                                                    #
 Activate nvm.
nvm install node                                                               #
 Use nvm to install npm (and Node.js at the same time).
```

ブラウザ JavaScript で の SDK を設定するには

HTML ページで の SDK を使用するには、 JavaScript WebPack を使用して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 1 つの JavaScript ファイルにバンドルし、HTML ページの のスクリプトタグに追加<head>します。例えば、

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

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

ステップ 4: サンプルコードをダウンロード

前のステップで開いたターミナルを使用して、SDK for のサンプルコードを AWS Cloud9 開発環境にダウンロード JavaScript します。(IDE でターミナルが開いていない場合は、IDE のメニューバーで [WindowWindow] (ウィンドウ)、[New Terminal] (新しいターミナル) の順に選択してターミナルを開きます。)

コード例のダウンロードには、次のコマンドを実行します。このコマンドは、公式 AWS SDK ドキュメントで使用されているすべてのコード例のコピーを環境のルートディレクトリにダウンロードします。

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

SDK for のコード例を見つけるには JavaScript、環境ウィンドウを使用して を開きます。*ENVIRONMENT_NAME*\aws-doc-sdk-examples\javascriptv3\example_code/srcの*ENVIRONMENT_NAME* は AWS Cloud9 開発環境の名前です。

これらのコード例やその他のコード例を操作する方法については、「 SDK for JavaScript code examples」を参照してください。

ステップ 5: サンプルコードを実行してデバッグする

AWS Cloud9 開発環境でコードを実行するには、「 ユーザーガイド」の「コードを実行する」を参照してください。 AWS Cloud9

Node.js コードをデバッグするには、AWS Cloud9 ユーザーガイドのコードのデバッグを参照してください。

SDK JavaScript (v3) コード例の

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 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。

サービス

- API for JavaScript (v3) SDK を使用したゲートウェイの例
- for JavaScript (v3) を使用した Aurora SDK の例
- for JavaScript (v3) を使用した Auto Scaling SDK の例
- for JavaScript (v3) を使用した Amazon Bedrock SDK の例
- SDK for JavaScript (v3) を使用した Amazon Bedrock ランタイムの例
- for JavaScript (v3) を使用した Agents SDK for Amazon Bedrock の例
- for JavaScript (v3) を使用した Agents SDK for Amazon Bedrock ランタイムの例
- CloudWatch JavaScript (v3) SDK に を使用する の例
- CloudWatch for JavaScript (v3) SDK を使用したイベントの例
- CloudWatch SDK for JavaScript (v3) を使用したログの例
- CodeBuild JavaScript (v3) SDK に を使用する の例
- for JavaScript (v3) を使用した Amazon Cognito ID SDK プロバイダーの例
- for JavaScript (v3) を使用した Amazon Comprehend SDK の例
- for JavaScript (v3) を使用した Amazon DocumentDB SDK の例
- JavaScript (v3) SDK用の を使用した DynamoDB の例
- for JavaScript (v3) SDK を使用した Amazon EC2の例
- Elastic Load Balancing - JavaScript (v3) SDK用の を使用したバージョン 2 の例
- EventBridge JavaScript (v3) SDK に を使用する の例

- AWS Glue JavaScript (v3) SDK に を使用する の例
- HealthImaging JavaScript (v3) SDK に を使用する の例
- IAM JavaScript (v3) SDK に を使用する の例
- for JavaScript (v3) を使用した SDK Kinesis の例
- for JavaScript (v3) を使用する Lambda SDK の例
- for JavaScript (v3) を使用した Amazon Lex の例 SDK
- for JavaScript (v3) SDK を使用した Amazon MSKの例
- for JavaScript (v3) を使用した Amazon Personalize SDK の例
- JavaScript (v3) SDK用の を使用した Amazon Personalize Events の例
- SDK for JavaScript (v3) を使用した Amazon Personalize ランタイムの例
- for JavaScript (v3) を使用した Amazon Pinpoint SDK の例
- JavaScript (v3) に を使用する Amazon Polly の例 SDK
- for JavaScript (v3) SDK を使用した Amazon RDSの例
- for JavaScript (v3) を使用した Amazon RDS Data Service SDK の例
- for JavaScript (v3) を使用した Amazon Redshift SDK の例
- for JavaScript (v3) を使用した Amazon Rekognition SDK の例
- JavaScript (v3) に を使用する Amazon S3 の例 SDK
- SDK を使用した S3 Glacier の例 JavaScript (v3)
- SageMaker JavaScript (v3) SDK に を使用する の例
- for JavaScript (v3) を使用した Secrets Manager SDK の例
- for JavaScript (v3) SDK を使用した Amazon SESの例
- for JavaScript (v3) SDK を使用した Amazon SNSの例
- for JavaScript (v3) SDK を使用した Amazon SQSの例
- JavaScript (v3) SDK に を使用する Step Functions の例
- AWS STS JavaScript (v3) SDK に を使用する の例
- AWS Support JavaScript (v3) SDK に を使用する の例
- JavaScript (v3) SDKに を使用する Amazon Textract の例
- for JavaScript (v3) を使用した Amazon Transcribe SDK の例
- for JavaScript (v3) を使用した Amazon Translate SDK の例

API for JavaScript (v3) SDK を使用したゲートウェイの例

次のコード例は、APIGateway 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の 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セットアップと実行の手順については、「」の詳細な例を参照してください GitHub。

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 ゲートウェイ
- 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 の JavaScript (v3)

Lambda JavaScript ランタイム を使用して AWS Lambda 関数を作成する方法を示しますAPI。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作業記念日を確認し、Amazon Simple Notification Service (Amazon SNS) を使用して、1 年間の記念日に従業員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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

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 ゲートウェイ
- DynamoDB
- Lambda
- Amazon SNS

for JavaScript (v3) を使用した Aurora SDK 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の 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) 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SES。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 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 を使用して、フィルタリングされた作業項目の E メールレポートを送信しますSES。
- 付属の 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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

for JavaScript (v3) を使用した Auto Scaling SDK 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

- API 詳細については、「 リファレンスAttachLoadBalancerTargetGroups」の「」を参照してください。 AWS SDK for JavaScript API

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、インスタンスの数を指定された範囲内に保持します。
- Elastic Load Balancing を使用してHTTPリクエストを処理し、配信します。 Elastic Load Balancing
- 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 "url";

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

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),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
```

```
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

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(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
```

```
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
```

```
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
```

```
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
```

```
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
```

```
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
```

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

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
```

```
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
```

```
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
```

```
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

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

```
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
```

```
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

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

- API 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

for JavaScript (v3) を使用した Amazon Bedrock SDK 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { fileURLToPath } from "url";

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

const REGION = "us-east-1";
const client = new BedrockClient({ region: REGION });

export const main = async () => {
  const command = new ListFoundationModelsCommand({});

  const response = await client.send(command);
  const models = response.modelSummaries;

  console.log("Listing the available Bedrock foundation models:");

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

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

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

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

  return response;
};

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

- API 詳細については、「 リファレンスListFoundationModels」の「」を参照してください。AWS SDK for JavaScript API

トピック

- アクション

アクション

`GetFoundationModel`

次のコード例は、GetFoundationModel を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

基盤モデルに関する詳細を取得します。

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

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

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

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

  const response = await client.send(command);

  return response.modelDetails;
};

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

- API 詳細については、「 リファレンスGetFoundationModel」の「」を参照してください。 AWS SDK for JavaScript API

`ListFoundationModels`

次のコード例は、`ListFoundationModels` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

使用可能な基盤モデルを一覧表示します。

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

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

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

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

  const command = new ListFoundationModelsCommand(input);
```

```
  const response = await client.send(command);

  return response.modelSummaries;
};

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

- API 詳細については、「 リファレンスListFoundationModels」の「」を参照してください。 AWS SDK for JavaScript API

SDK for JavaScript (v3) を使用した 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

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

const AWS_REGION = "us-east-1";

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

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

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

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

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

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

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

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

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

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

- API 詳細については、「 リファレンスInvokeModel」の「」を参照してください。 AWS SDK for JavaScript API

トピック

- シナリオ
- AI21 ラボJurassic-2
- Amazon Titan Text
- Anthropic Claude

- Cohere Command
- メタラマ
- ミスラル 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) でプロンプトを準備して送信する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { fileURLToPath } from "url";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { FoundationModels } from "../config/foundation_models.js";

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

const greeting = new ScenarioOutput(
  "greeting",
  "Welcome to the Amazon Bedrock Runtime client demo!",
```

```
  { header: true },
);

const selectModel = new ScenarioInput("model", "First, select a model:", {
  type: "select",
  choices: Object.values(FoundationModels).map((model) => ({
    name: model.modelName,
    value: model,
  })),
});

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

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

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

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

```
const scenario = new Scenario("Amazon Bedrock Runtime Demo", [
  greeting,
  selectModel,
  enterPrompt,
  printDetails,
  invokeModel,
  printResponse,
]);

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

- API 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

AI21 ラボJurassic-2

会話

次のコード例は、Bedrock の Converse を使用して AI21 Labs Jurassic-2 にテキストメッセージを送信する方法を示していますAPI。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して、AI21Labs Jurassic-2 にテキストメッセージを送信しますAPI。

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

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

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

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

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

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

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

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

- API 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「会話」を参照してください。

InvokeModel

次のコード例は、Invoke Model を使用して AI21 Labs Jurassic-2 にテキストメッセージを送信する方法を示していますAPI。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

モデル呼び出しを使用してテキストメッセージAPIを送信します。

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

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

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

/**
 * Invokes an AI21 Labs Jurassic-2 model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
```

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

  // Prepare the payload for the model.
  const payload = {
    prompt,
    maxTokens: 500,
    temperature: 0.5,
  };

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

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

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

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

```
}
```

- API 詳細については、「 リファレンスInvokeModel」の「」を参照してください。 AWS SDK for JavaScript API

Amazon Titan Text

会話

次のコード例は、Bedrock の Converse 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API。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して、Amazon Titan Text にテキストメッセージを送信しますAPI。

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

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

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

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

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
```

```
    role: "user",
    content: [{ text: userMessage }],
  },
];

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

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

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

- API 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「会話」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse を使用して Amazon Titan Text にテキストメッセージを送信API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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して Amazon Titan Text にテキストメッセージを送信API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 詳細については、「 リファレンスConverseStream」の「」を参照してください。 AWS SDK for JavaScript API

InvokeModel

次のコード例は、Invoke Model 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API。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

モデル呼び出しを使用してテキストメッセージAPIを送信します。

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

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

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

/**
 * Invokes an Amazon Titan Text generation model.
 *
```

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

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

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

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

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

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

- API 詳細については、「 リファレンスInvokeModel」の「」を参照してください。 AWS SDK for JavaScript API

Anthropic Claude

会話

次のコード例は、Bedrock の Converse 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API。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して、Anthropic Claude にテキストメッセージを送信しますAPI。

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

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

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

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

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

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

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

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

- API 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「会話」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse を使用して Anthropic Claude にテキストメッセージを送信API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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して Anthropic Claude にテキストメッセージを送信API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 詳細については、「 リファレンスConverseStream」の「」を参照してください。 AWS SDK for JavaScript API

InvokeModel

次のコード例は、Invoke Model 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API。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

モデル呼び出しを使用してテキストメッセージAPIを送信します。

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

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
```

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

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

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

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
```

```
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

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

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

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

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
```

```
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

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

  let completeMessage = "";

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

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

  // Return the final response
  return completeMessage;
};

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

  try {
    console.log("-".repeat(53));
```

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

- API 詳細については、「 リファレンスInvokeModel」の「」を参照してください。 AWS SDK for JavaScript API

InvokeModelWithResponseStream

次のコード例は、Invoke Model を使用して Anthropic Claude モデルにテキストメッセージを送信し API、レスポンスストリームを出力する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Invoke Model を使用してテキストメッセージAPI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 "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 詳細については、「 リファレンスInvokeModelWithResponseStream」の「」を参照してください。 AWS SDK for JavaScript API

Cohere Command

会話

次のコード例は、Bedrock の Converse 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API。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して、Cohere コマンドにテキストメッセージを送信しますAPI。

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

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

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

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

// Start a conversation with the user message.
const userMessage =
```

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

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

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

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

- API 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「会話」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse を使用してテキストメッセージを Cohere Command に送信API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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して Cohere Command にテキストメッセージを送信API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 詳細については、「 リファレンスConverseStream」の「」を参照してください。 AWS SDK for JavaScript API

メタラマ

会話

次のコード例は、Bedrock の Converse 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API。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して、Meta Llama にテキストメッセージを送信しますAPI。

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

import {
  BedrockRuntimeClient,
  ConverseCommand,
```

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

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

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

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

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

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

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

- API 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「会話」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse を使用してテキストメッセージを Meta Llama に送信API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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して Meta Llama にテキストメッセージを送信API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 詳細については、「 リファレンスConverseStream」の「」を参照してください。 AWS SDK for JavaScript API

InvokeModel: ラマ 2

次のコード例は、モデル を呼び出す を使用して Meta Llama 2 にテキストメッセージを送信する方法を示していますAPI。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

モデル呼び出しを使用してテキストメッセージAPIを送信します。

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

import {
```

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

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

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

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

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

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

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

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

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

// Learn more about the Llama 2 prompt format at:
```

```
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-2
```

- API 詳細については、「 リファレンスInvokeModel」の「」を参照してください。 AWS SDK for JavaScript API

InvokeModel: ラマ 3

次のコード例は、モデル を呼び出す を使用して Meta Llama 3 にテキストメッセージを送信する方法を示していますAPI。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

モデル呼び出しを使用してテキストメッセージAPI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 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

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

// Embed the message in Llama 3's prompt format.
const prompt = `
```

```
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

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

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

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

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

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

- API 詳細については、「 リファレンスInvokeModel」の「」を参照してください。 AWS SDK for JavaScript API

InvokeModelWithResponseStream: ラマ 2

次のコード例は、モデル を呼び出し、レスポンスストリームをAPI出力して、Meta Llama 2 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Invoke Model を使用してテキストメッセージAPI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 2 and print the response stream in real-time.

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

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

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

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

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

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

```
  top_p: 0.9,
};

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

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

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

- API 詳細については、「 リファレンスInvokeModelWithResponseStream」の「」を参照してください。 AWS SDK for JavaScript API

InvokeModelWithResponseStream: ラマ 3

次のコード例は、モデル を呼び出し、レスポンスストリームをAPI出力して、Meta Llama 3 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Invoke Model を使用してテキストメッセージAPI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 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

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

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

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

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

```
    modelId,
  }),
);

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

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

- API 詳細については、「 リファレンスInvokeModelWithResponseStream」の「」を参照してください。 AWS SDK for JavaScript API

ミスラル AI

会話

次のコード例は、Bedrock の Converse 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API。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して、Mistral にテキストメッセージを送信しますAPI。

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

import {
```

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

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

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

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

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

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

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

- API 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「会話」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse を使用してテキストメッセージを Mistral に送信API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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Bedrock の Converse を使用して Mistral にテキストメッセージを送信API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 詳細については、「 リファレンスConverseStream」の「」を参照してください。 AWS SDK for JavaScript API

InvokeModel

次のコード例は、モデル を呼び出す を使用して、ミスラル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API。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

呼び出しモデルを使用してテキストメッセージAPIを送信します。

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

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

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

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

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `<s>[INST] ${prompt} [/INST]`;

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

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

```
  });
  const apiResponse = await client.send(command);

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

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

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

- API 詳細については、「 リファレンスInvokeModel」の「」を参照してください。 AWS SDK for JavaScript API

for JavaScript (v3) を使用した Agents SDK for Amazon Bedrock の例

次のコード例は、 Agents for 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

開始方法

Amazon Bedrock の Hello Agents

次のコード例は、Agents for Amazon Bedrock の使用を開始する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { fileURLToPath } from "url";

import {
  BedrockAgentClient,
  GetAgentCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * @typedef {Object} AgentSummary
 */

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 region.
 * It then retrieves a list of existing agents using the streamlined paginator
 approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
 * - Use of the Bedrock Agents client to initialize and communicate with the AWS
 service.
```

```
 * - Listing resources in a paginated response pattern.
 * - Accessing an individual resource using a command object.
 *
 * @returns {Promise<void>} A promise that resolves when the function has completed
 execution.
 */
export const main = async () => {
  const region = "us-east-1";

  console.log("=".repeat(68));

  console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
  const client = new BedrockAgentClient({ region });

  console.log(`Retrieving the list of existing agents...`);
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});

  /** @type {AgentSummary[]} */
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  console.log(`Found ${agentSummaries.length} agents in ${region}.`);

  if (agentSummaries.length > 0) {
    for (const agentSummary of agentSummaries) {
      const agentId = agentSummary.agentId;
      console.log("=".repeat(68));
      console.log(`Retrieving agent with ID: ${agentId}:`);
      console.log("-".repeat(68));

      const command = new GetAgentCommand({ agentId });
      const response = await client.send(command);
      const agent = response.agent;

      console.log(` Name: ${agent.agentName}`);
      console.log(` Status: ${agent.agentStatus}`);
      console.log(` ARN: ${agent.agentArn}`);
      console.log(` Foundation model: ${agent.foundationModel}`);
    }
  }
  console.log("=".repeat(68));
```

```
};

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

- API 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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

エージェントを作成します。

```
import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  CreateAgentCommand,
```

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

/**
 * Creates an Amazon Bedrock Agent.
 *
 * @param {string} agentName - A name for the agent that you create.
 * @param {string} foundationModel - The foundation model to be used by the agent
 you create.
 * @param {string} agentResourceRoleArn - The ARN of the IAM role with permissions
 required by the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing details of the created agent.
 */
export const createAgent = async (
  agentName,
  foundationModel,
  agentResourceRoleArn,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  const command = new CreateAgentCommand({
    agentName,
    foundationModel,
    agentResourceRoleArn,
  });
  const response = await client.send(command);

  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentName and accountId, and roleName with a
 unique name for the new agent,
  // the id of your AWS account, and the name of an existing execution role that the
 agent can use inside your account.
  // For foundationModel, specify the desired model. Ensure to remove the brackets
 '[]' before adding your data.

  // A string (max 100 chars) that can include letters, numbers, dashes '-', and
 underscores '_'.
  const agentName = "[your-bedrock-agent-name]";
```

```
  // Your AWS account id.
  const accountId = "[123456789012]";

  // The name of the agent's execution role. It must be prefixed by
 `AmazonBedrockExecutionRoleForAgents_`.
  const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

  // The ARN for the agent's execution role.
  // Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
  const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

  // Specify the model for the agent. Change if a different model is preferred.
  const foundationModel = "anthropic.claude-v2";

  // Check for unresolved placeholders in agentName and roleArn.
  checkForPlaceholders([agentName, roleArn]);

  console.log(`Creating a new agent...`);

  const agent = await createAgent(agentName, foundationModel, roleArn);
  console.log(agent);
}
```

- API 詳細については、「 リファレンスCreateAgent」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteAgent`

次のコード例は、`DeleteAgent` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

エージェントを削除します。

```
import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent to delete.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-
agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
 and some additional metadata.
 */
export const deleteAgent = (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });
  const command = new DeleteAgentCommand({ agentId });
  return client.send(command);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets (`[]`) before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Deleting agent with ID ${agentId}...`);

  const response = await deleteAgent(agentId);
  console.log(response);
}
```

- API 詳細については、「 リファレンスDeleteAgent」の「」を参照してください。 AWS SDK for JavaScript API

`GetAgent`

次のコード例は、`GetAgent` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

エージェントを取得します。

```
import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing the agent details.
 */
export const getAgent = async (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const command = new GetAgentCommand({ agentId });
  const response = await client.send(command);
  return response.agent;
};
```

```
// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Retrieving agent with ID ${agentId}...`);

  const agent = await getAgent(agentId);
  console.log(agent);
}
```

- API 詳細については、「 リファレンスGetAgent」の「」を参照してください。 AWS SDK for JavaScript API

`ListAgentActionGroups`

次のコード例は、ListAgentActionGroups を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 "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 詳細については、「 リファレンスListAgentActionGroups」の「」を参照してください。AWS SDK for JavaScript API

`ListAgents`

次のコード例は、`ListAgents` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 "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 詳細については、「 リファレンスListAgents」の「」を参照してください。 AWS SDK for JavaScript API

for JavaScript (v3) を使用した Agents SDK for Amazon Bedrock ランタイムの例

次のコード例は、 Agents for 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。

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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

- アクション

アクション

`InvokeAgent`

次のコード例は、InvokeAgent を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  BedrockAgentRuntimeClient,
  InvokeAgentCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
  // });
```

```
  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (let chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
      console.log(chunk);
      const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
      completion += decodedResponse;
    }

    return { sessionId: sessionId, completion };
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- API 詳細については、「 リファレンスInvokeAgent」の「」を参照してください。 AWS SDK for JavaScript API

CloudWatch JavaScript (v3) SDK に を使用する 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 CloudWatch。

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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });
```

```
  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

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、「 リファレンスDeleteAlarms」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};
```

```
cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスDeleteAlarms」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeAlarmsForMetric`

次のコード例は、`DescribeAlarmsForMetric` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
```

```
  }
};

export default run();
```

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、「 リファレンスDescribeAlarmsForMetric」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
```

```
  }
});
```

- 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、「 リファレンスDescribeAlarmsForMetric」の「」を参照してください。AWS SDK for JavaScript API

`DisableAlarmActions`

次のコード例は、`DisableAlarmActions` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

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、「 リファレンスDisableAlarmActions」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 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、「 リファレンスDisableAlarmActions」の「」を参照してください。 AWS SDK for JavaScript API

`EnableAlarmActions`

次の例は、`EnableAlarmActions` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

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、「 リファレンスEnableAlarmActions」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
```

```
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

- 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、「 リファレンスEnableAlarmActions」の「」を参照してください。 AWS SDK for JavaScript API

`ListMetrics`

次のコード例は、`ListMetrics` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";
```

```
export const main = () => {
  // Use the AWS console to see available namespaces and metric names. Custom
 metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
    Dimensions: [
      {
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

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

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、「 リファレンスListMetrics」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- 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、「 リファレンスListMetrics」の「」を参照してください。 AWS SDK for JavaScript API

`PutMetricAlarm`

次の例は、`PutMetricAlarm` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
 the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

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、「 リファレンスPutMetricAlarm」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};
```

```
cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスPutMetricAlarm」の「」を参照してください。 AWS SDK for JavaScript API

`PutMetricData`

次のコード例は、PutMetricData を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
```

```
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

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、「 リファレンスPutMetricData」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

- 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、「 リファレンスPutMetricData」の「」を参照してください。 AWS SDK for JavaScript API

CloudWatch for JavaScript (v3) SDK を使用したイベントの例

次のコード例は、 CloudWatch イベントで 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

- アクション

アクション

`PutEvents`

次のコード例は、`PutEvents` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
```

```
        Source: "my.app",

        // The name of the event that is being sent.
        DetailType: "My Custom Event",

        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

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 詳細については、「 リファレンスPutEvents」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

cwevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- 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、「 リファレンスPutEvents」の「」を参照してください。 AWS SDK for JavaScript API

`PutRule`

次のコード例は、`PutRule` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  // Request parameters for PutRule.
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/
API_PutRule.html#API_PutRule_RequestParameters
  const command = new PutRuleCommand({
    Name: process.env.CLOUDWATCH_EVENTS_RULE,

    // The event pattern for the rule.
    //  Example: {"source": ["my.app"]}
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,

    // The state of the rule. Valid values: ENABLED, DISABLED
    State: "ENABLED",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

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 詳細については、「 リファレンスPutRule」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

cwevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- 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、「 リファレンスPutRule」の「」を参照してください。 AWS SDK for JavaScript API

`PutTargets`

次のコード例は、`PutTargets` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutTargetsCommand({
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,

    // The targets to add to the rule.
    Targets: [
      {
        Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
        // The ID of the target. Choose a unique ID for each target.
        Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

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 詳細については、「 リファレンスPutTargets」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myCloudWatchEventsTarget",
    },
  ],
};

cwevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスPutTargets」の「」を参照してください。 AWS SDK for JavaScript API

CloudWatch SDK for JavaScript (v3) を使用したログの例

次のコード例は、 CloudWatch ログで 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

アクション

`CreateLogGroup`

次のコード例は、`CreateLogGroup` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスCreateLogGroup」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteLogGroup`

次の例は、DeleteLogGroup を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteLogGroup」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteSubscriptionFilter`

次の例は、`DeleteSubscriptionFilter` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteSubscriptionFilter」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスDeleteSubscriptionFilter」の「」を参照してください。 AWS SDK for JavaScript API

DescribeLogGroups

次の例は、DescribeLogGroups を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups && page.logGroups.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }

  console.log(logGroups);
  return logGroups;
};
```

- API 詳細については、「 リファレンスDescribeLogGroups」の「」を参照してください。AWS SDK for JavaScript API

DescribeSubscriptionFilters

次のコード例は、DescribeSubscriptionFilters を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeSubscriptionFilters」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cwl.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

- 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、「 リファレンスDescribeSubscriptionFilters」の「」を参照してください。 AWS SDK for JavaScript API

`GetQueryResults`

次の例は、`GetQueryResults` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
```

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

- API 詳細については、「 リファレンスGetQueryResults」の「」を参照してください。 AWS SDK for JavaScript API

`PutSubscriptionFilter`

次の例は、`PutSubscriptionFilter` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスPutSubscriptionFilter」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスPutSubscriptionFilter」の「」を参照してください。 AWS SDK for JavaScript API

`StartLiveTail`

次のコード例は、`StartLiveTail` を使用する方法を示しています。

SDK JavaScript (v3) 用の

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 詳細については、「 リファレンスStartLiveTail」の「」を参照してください。 AWS SDK for JavaScript API

`StartQuery`

次の例は、StartQuery を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
/**
 * 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 詳細については、「 リファレンスStartQuery」の「」を参照してください。 AWS SDK for JavaScript API

シナリオ

大規模なクエリを実行する

次のコード例は、 CloudWatch ログを使用して 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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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(parseInt(process.env.QUERY_START_DATE)),
    new Date(parseInt(process.env.QUERY_END_DATE)),
  ],
});

await cloudWatchQuery.run();

console.log(
```

```
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found:
 ${cloudWatchQuery.results.length}`,
);
```

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 [];
      } else {
        throw err;
      }
    }
  }

  /**
   * Wrapper for the StartQueryCommand. Uses a static query string
   * for consistency.
   * @param {[Date, Date]} dateRange
   * @param {number} maxLogs
   * @returns {Promise<{ queryId: string }>}
   */
  async _startQuery([startDate, endDate], maxLogs = 10000) {
    try {
      return await this.client.send(
        new StartQueryCommand({
          logGroupNames: this.logGroupNames,
          queryString: "fields @timestamp, @message | sort @timestamp asc",
          startTime: startDate.valueOf(),
          endTime: endDate.valueOf(),
          limit: maxLogs,
        }),
      );
    } catch (err) {
      /** @type {string} */
      const message = err.message;
      if (message.startsWith("Query's end date and time")) {
        // This error indicates that the query's start or end date occur
        // before the log group was created.
        throw new DateOutOfBoundsError(message);
      }

      throw err;
```

```
    }
  }

  /**
   * Call GetQueryResultsCommand until the query is done.
   * @param {string} queryId
   */
  _waitUntilQueryDone(queryId) {
    const getResults = async () => {
      const results = await this._getQueryResults(queryId);
      const queryDone = [
        "Complete",
        "Failed",
        "Cancelled",
        "Timeout",
        "Unknown",
      ].includes(results.status);

      return { queryDone, results };
    };

    return retry(
      { intervalInMs: 1000, maxRetries: 60, quiet: true },
      async () => {
        const { queryDone, results } = await getResults();
        if (!queryDone) {
          throw new Error("Query not done.");
        }

        return results;
      },
    );
  }
}
```

- API 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

CodeBuild JavaScript (v3) SDK に を使用する 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 CodeBuild。

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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスCreateProject」の「」を参照してください。 AWS SDK for JavaScript API

for JavaScript (v3) を使用した Amazon Cognito ID SDK 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
```

```
  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 詳細については、「 リファレンスListUserPools」の「」を参照してください。 AWS SDK for JavaScript API

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

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

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

- API 詳細については、「 リファレンスAdminGetUser」の「」を参照してください。 AWS SDK for JavaScript API

`AdminInitiateAuth`

次の例は、AdminInitiateAuth を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスAdminInitiateAuth」の「」を参照してください。 AWS SDK for JavaScript API

`AdminRespondToAuthChallenge`

次のコード例は、AdminRespondToAuthChallenge を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスAdminRespondToAuthChallenge」の「」を参照してください。 AWS SDK for JavaScript API

`AssociateSoftwareToken`

次の例は、AssociateSoftwareToken を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

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

- API 詳細については、「 リファレンスAssociateSoftwareToken」の「」を参照してください。 AWS SDK for JavaScript API

`ConfirmDevice`

次の例は、ConfirmDevice を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスConfirmDevice」の「」を参照してください。 AWS SDK for JavaScript API

`ConfirmSignUp`

次の例は、`ConfirmSignUp` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const confirmSignUp = ({ clientId, username, code }) => {
```

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

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

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

- API 詳細については、「 リファレンスConfirmSignUp」の「」を参照してください。 AWS SDK for JavaScript API

`InitiateAuth`

次の例は、`InitiateAuth` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスInitiateAuth」の「」を参照してください。 AWS SDK for JavaScript API

`ListUsers`

次の例は、`ListUsers` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

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

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

- API 詳細については、「 リファレンスListUsers」の「」を参照してください。 AWS SDK for JavaScript API

`ResendConfirmationCode`

次のコード例は、`ResendConfirmationCode` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

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

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

- API 詳細については、「 リファレンスResendConfirmationCode」の「」を参照してください。 AWS SDK for JavaScript API

`RespondToAuthChallenge`

次の例は、RespondToAuthChallenge を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスRespondToAuthChallenge」の「」を参照してください。 AWS SDK for JavaScript API

`SignUp`

次の例は、SignUp を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスSignUp」の「」を参照してください。 AWS SDK for JavaScript API

`VerifySoftwareToken`

次のコード例は、`VerifySoftwareToken` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスVerifySoftwareToken」の「」を参照してください。AWS SDK for JavaScript API

シナリオ

を必要とするユーザープールでユーザーをサインアップする MFA

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ることで、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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

最良のエクスペリエンスを得るには、 GitHub リポジトリのクローンを作成し、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 { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};
```

```
const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up'
 command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Signing up.`);
    await signUp({ clientId, username, password, email });
    log(`Signed up. A confirmation email has been sent to: ${email}.`);
    log(`Run 'confirm-sign-up ${username} <code>' to confirm your account.`);
  } catch (err) {
    log(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

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

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-
sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
```

```
    log(`Confirming user.`);
    await confirmSignUp({ clientId, username, code });
    log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
    );
  } catch (err) {
    log(err);
  }
};

export { confirmSignUpHandler };

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

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

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../../actions/associate-software-
token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-
token' to finish the setup.",
  );
  qrcode.generate(
```

```
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-initiate-
auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);
```

```
    log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    log(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../actions/admin-respond-to-auth-
challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
```

```
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-
respond-to-auth-challenge' command.`,
    );
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an
 argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    log("Successfully authenticated.");
  } catch (err) {
    log(err);
```

```
  }
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-
software-token' command.`,
    );
  }
};
const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;

  try {
    validateTotp(totp);
```

```
    log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    console.log(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

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

- API 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

for JavaScript (v3) を使用した Amazon Comprehend SDK 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 JavaScript (v3) 用の

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、文字起こし、翻訳し、Amazon Simple Email Service (Amazon) 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を示しますSES。

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、「」の詳細な例を参照してくださいGitHub。

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 JavaScript (v3) 用の

Amazon Lex を使用してウェブアプリケーション内に Chatbot APIを作成し、ウェブサイトの訪問者をエンゲージさせる方法を示します。

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、 AWS SDK for JavaScript デ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 JavaScript (v3) 用の

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、「」の「 GitHub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

for JavaScript (v3) を使用した Amazon DocumentDB SDK 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

- サーバーレスサンプル

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、レコードの内容をログ記録します。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で Amazon DocumentDB イベントを消費する JavaScript。

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

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

を使用した Lambda での Amazon DocumentDB イベントの消費 TypeScript

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

```
console.log('Loading function');

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

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

JavaScript (v3) SDK用の 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

での DynamoDB の使用の詳細については AWS SDK for JavaScript、「 を使用した DynamoDB のプログラミング JavaScript」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 詳細については、「 リファレンスListTables」の「」を参照してください。 AWS SDK for JavaScript API

トピック

- 基礎
- アクション
- シナリオ
- サーバーレスサンプル

基礎

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ファイルからテーブルに映画データを書き込みます。
- 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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { readFileSync } from "fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";
```

```
// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

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

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
```

```
    // The way your data is accessed determines how you structure your keys.
    // The movies table will be queried for movies by year. It makes sense
    // to make year our partition (HASH) key.
    { AttributeName: "year", KeyType: "HASH" },
    { AttributeName: "title", KeyType: "RANGE" },
  ],
});

log("Creating a table.");
const createTableResponse = await client.send(createTableCommand);
log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

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

/**
 * Add a movie to the table.
 */

log("Adding a single movie to the table.");
// PutCommand is the first example usage of 'lib-dynamodb'.
const putCommand = new PutCommand({
  TableName: tableName,
  Item: {
    // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
1981 }`)
    // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
    year: 1981,
    // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
    // is required.
    title: "The Evil Dead",
    // Every other attribute is optional.
    info: {
      genres: ["Horror"],
    },
  },
});
await docClient.send(putCommand);
log("The movie was added.");

/**
```

```
    * Get a movie from the table.
    */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
```

```
 */

log("Deleting a single movie from the table.");
const deleteCommand = new DeleteCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
});
await client.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
}
log("Movies added.");

/**
 * Query for movies by year.
 */
```

```
  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log(`Scan for movies released between 1980 and 1990`);
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression. Scan
 will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
```

```
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

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

- API 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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスBatchExecuteStatement」の「」を参照してください。AWS SDK for JavaScript API

`BatchGetItem`

次のコード例は、BatchGetItem を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「」を参照してくださいBatchGet。

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

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

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
```

```
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses["Books"]);
  return response;
};
```

- 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、「 リファレンスBatchGetItem」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
```

```
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- 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、「 リファレンスBatchGetItem」の「」を参照してください。 AWS SDK for JavaScript API

`BatchWriteItem`

次の例は、`BatchWriteItem` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「」を参照してくださいBatchWrite。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

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

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year' is
 recommended
        // to account for duplicate titles.
        ["BatchWriteMoviesTable"]: putRequests,
```

```
      },
    });

    await docClient.send(command);
  }
};
```

- API 詳細については、「 リファレンスBatchWriteItem」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
```

```
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスBatchWriteItem」の「」を参照してください。 AWS SDK for JavaScript API

`CreateTable`

次の例は、CreateTable を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  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 詳細については、「 リファレンスCreateTable」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
```

```
  }
});
```

- 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、「 リファレンスCreateTable」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteItem`

次のコード例は、`DeleteItem` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「」を参照してくださいDeleteCommand。

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

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

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

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

```
};
```

- 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、「 リファレンスDeleteItem」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

テーブルから項目を削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルから項目を削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスDeleteItem」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteTable`

次の例は、`DeleteTable` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteTable」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
```

```
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスDeleteTable」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeTable`

次の例は、`DescribeTable` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 詳細については、「 リファレンスDescribeTable」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- 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、「 リファレンスDescribeTable」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeTimeToLive`

次のコード例は、`DescribeTimeToLive` を使用する方法を示しています。

SDK JavaScript (v3) 用の

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

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

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

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

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

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

- API 詳細については、「 リファレンスDescribeTimeToLive」の「」を参照してください。 AWS SDK for JavaScript API

`ExecuteStatement`

次のコード例は、ExecuteStatement を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスExecuteStatement」の「」を参照してください。 AWS SDK for JavaScript API

`GetItem`

次のコード例は、`GetItem` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「」を参照してくださいGetCommand。

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

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

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

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

- API 詳細については、「 リファレンスGetItem」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

テーブルから項目を取得します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
```

```
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルから項目を取得します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- 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、「 リファレンスGetItem」の「」を参照してください。 AWS SDK for JavaScript API

`ListTables`

次の例は、`ListTables` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 詳細については、「 リファレンスListTables」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- 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、「 リファレンスListTables」の「」を参照してください。 AWS SDK for JavaScript API

`PutItem`

次のコード例は、`PutItem` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「」を参照してくださいPutCommand。

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

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

```
export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

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

- API 詳細については、「 リファレンスPutItem」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

テーブルに項目を配置します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};
```

```
// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルに項目を配置します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスPutItem」の「」を参照してください。 AWS SDK for JavaScript API

`Query`

次の例は、`Query` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「」を参照してくださいQueryCommand。

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

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

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

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

- 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、「 リファレンス」の「クエリ」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- 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、「 リファレンス」の「クエリ」を参照してください。 AWS SDK for JavaScript API

Scan

次のコード例は、Scan を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「」を参照してくださいScanCommand。

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

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

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- API 詳細については、「 リファレンス」の「スキャン」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
```

```
  }
});
```

- 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、「 リファレンス」の「スキャン」を参照してください。 AWS SDK for JavaScript API

`UpdateItem`

次の例は、UpdateItem を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「」を参照してくださいUpdateCommand。

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

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

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
```

```
  });

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

- API 詳細については、「 リファレンスUpdateItem」の「」を参照してください。 AWS SDK for JavaScript API

`UpdateTimeToLive`

次のコード例は、UpdateTimeToLive を使用する方法を示しています。

SDK の JavaScript (v3)

既存の DynamoDB テーブルTTLで を有効にします。

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

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

    const client = new DynamoDBClient({});

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

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

```
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

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

既存の DynamoDB テーブルTTLで を無効にします。

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

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

    const client = new DynamoDBClient({});

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

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

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

- API 詳細については、「 リファレンスUpdateTimeToLive」の「」を参照してください。 AWS SDK for JavaScript API

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 の 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) 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ケーションを構築する方法を示しますSNS。

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、「」の詳細な例を参照してくださいGitHub。

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

項目の を条件付きで更新する TTL

次のコード例は、項目の を条件付きで更新する方法を示していますTTL。

SDK の JavaScript (v3)

条件を指定して、テーブル内の既存の DynamoDB 項目で TTLを更新します。

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

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

```
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

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

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

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

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

- API 詳細については、「 リファレンスUpdateItem」の「」を参照してください。 AWS SDK for JavaScript API

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 の 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セットアップと実行の手順については、「」の詳細な例を参照してください GitHub。

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 ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

DynamoDB データを追跡するウェブアプリケーションを作成する

次のコード例は、Amazon DynamoDB テーブル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 の JavaScript (v3)

Amazon DynamoDB を使用して、DynamoDB の作業データを追跡する動的ウェブアプリケーションAPI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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB

- Amazon SES

を使用して項目を作成する TTL

次のコード例は、 を使用して項目を作成する方法を示していますTTL。

SDK の JavaScript (v3)

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

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

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

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

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

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

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

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

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

- API 詳細については、「 リファレンスPutItem」の「」を参照してください。 AWS SDK for JavaScript API

イメージPPE内の検出

次のコード例は、Amazon Rekognition を使用して画像内の個人用保護具 (PPE) を検出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 の JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Simple Storage Service (Amazon S3PPE) バケットにあるイメージ内の個人用保護具 () を検出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。Amazon S3 アプリケーションは、結果を Amazon DynamoDB テーブルに保存し、Amazon Simple Email Service (Amazon) 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SES。

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 PPEを使用して のイメージを分析します。 Amazon Rekognition
- Amazon の E メールアドレスを確認しますSES。
- 結果で DynamoDB テーブルを更新します。
- Amazon を使用して E メール通知を送信しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

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 の JavaScript (v2)

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。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Lambda

SDK の 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 AWS SDK for JavaScript v3 を使用します。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Lambda

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します。
- 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します。
- 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します。
- 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します。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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?`,
      { confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

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

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

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

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

  /**
   * Insert items.
   */

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

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

  /**
   * Select items.
   */

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

  /**
   * Update items.
   */

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

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

  /**
   * Delete the items.
   */

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

  /**
   * Delete the table.
   */

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

- API 詳細については、「 リファレンスBatchExecuteStatement」の「」を参照してください。 AWS SDK for JavaScript API

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します。
- INSERT ステートメントを実行して項目を追加します。
- UPDATE ステートメントを実行して項目を更新します。
- DELETE ステートメントを実行して項目を削除します。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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?`,
      { 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 詳細については、「 リファレンスExecuteStatement」の「」を参照してください。 AWS SDK for JavaScript API

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 JavaScript (v3) 用の

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

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

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

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

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

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

- API 詳細については、「 リファレンス」の「クエリ」を参照してください。 AWS SDK for JavaScript API

項目の を更新する TTL

次のコード例は、項目の を更新する方法を示していますTTL。

SDK JavaScript (v3) 用の

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

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

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

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
```

```
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

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

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

- API 詳細については、「 リファレンスUpdateItem」の「」を参照してください。 AWS SDK for JavaScript API

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で DynamoDB イベントを消費する JavaScript。

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

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

を使用して Lambda で DynamoDB イベントを消費する TypeScript。

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

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用した Lambda での DynamoDB バッチアイテムの失敗のレポート JavaScript。

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

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

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

を使用した Lambda での DynamoDB バッチアイテムの失敗のレポート TypeScript。

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

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

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

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

for JavaScript (v3) SDK を使用した Amazon EC2の例

次のコード例は、Amazon 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EC2。

「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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

開始方法

Amazon へようこそ EC2

次のコード例は、Amazon の使用を開始する方法を示していますEC2。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  const client = new EC2Client();
  try {
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({}),
    );

    const securityGroupList = SecurityGroups.slice(0, 9)
      .map((sg) => ` • ${sg.GroupId}: ${sg.GroupName}`)
      .join("\n");

    console.log(
      "Hello, Amazon EC2! Let's list up to 10 of your security groups:",
    );
    console.log(securityGroupList);
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- API 詳細については、「 リファレンスDescribeSecurityGroups」の「」を参照してください。AWS SDK for JavaScript API

トピック

- 基礎
- アクション
- シナリオ

基礎

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。
- を使用してインスタンスに接続しSSH、リソースをクリーンアップします。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

このファイルには、 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EC2。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) {
      page.Parameters.forEach((param) => {
        // Filter by Amazon Linux 2
        if (param.Name.includes("amzn2")) {
          AMIs.push(param.Value);
        }
      });
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidFilterValue") {
      caught.message = `${caught.message} Please provide a valid filter value for
paginateGetParametersByPath.`;
    }
    state.errors.push(caught);
    return;
  }

  const imageDetails = [];
  const describeImagesPaginator = paginateDescribeImages(
    { client: state.ec2Client },
    // The images found from the call to SSM.
    { ImageIds: AMIs },
  );

  try {
    // Get more details for the images found above.
```

```
      for await (const page of describeImagesPaginator) {
        imageDetails.push(...(page.Images || []));
      }

      // Store the image details for later use.
      state["images"] = imageDetails;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidAMIID.NotFound") {
        caught.message = `${caught.message}. Please provide a valid image id.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const provideImage = new ScenarioInput(
  "image",
  "Select one of the following images.",
  {
    type: "select",
    choices: (/** @type { State } */ state) =>
      state.images.map((image) => ({
        name: `${image.ImageId} - ${image.Description}`,
        value: image,
      })),
    default: (/** @type { State } */ state) => state.images[0],
    skipWhen: skipWhenErrors,
  },
);

export const getCompatibleInstanceTypes = new ScenarioAction(
  "getCompatibleInstanceTypes",
  async (/** @type {State} */ state) => {
    // Get more details about instance types that match the architecture of
    // the provided image.
    const paginator = paginateDescribeInstanceTypes(
      { client: state.ec2Client, pageSize: 25 },
      {
        Filters: [
          {
            Name: "processor-info.supported-architecture",
            // The value selected from provideImage()
```

```
            Values: [state.image.Architecture],
          },
          // Filter for smaller, less expensive, types.
          { Name: "instance-type", Values: ["*.micro", "*.small"] },
        ],
      },
    );

    const instanceTypes = [];

    try {
      for await (const page of paginator) {
        if (page.InstanceTypes.length) {
          instanceTypes.push(...(page.InstanceTypes || []));
        }
      }

      if (!instanceTypes.length) {
        state.errors.push(
          "No instance types matched the instance type filters.",
        );
      }
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check the provided values and
 try again.`;
      }

      state.errors.push(caught);
    }

    state["instanceTypes"] = instanceTypes;
  },
  { skipWhen: skipWhenErrors },
);

export const provideInstanceType = new ScenarioInput(
  "instanceType",
  "Select an instance type.",
  {
    choices: (/** @type {State} */ state) =>
      state.instanceTypes.map((instanceType) => ({
        name: `${instanceType.InstanceType} - Memory:
${instanceType.MemoryInfo.SizeInMiB}`,
```

```
        value: instanceType.InstanceType,
      })),
    type: "select",
    default: (/** @type {State} */ state) =>
      state.instanceTypes[0].InstanceType,
    skipWhen: skipWhenErrors,
  },
);

export const runInstance = new ScenarioAction(
  "runInstance",
  async (/** @type { State } */ state) => {
    const { Instances } = await state.ec2Client.send(
      new RunInstancesCommand({
        KeyName: state[provideKeyPairName.name],
        SecurityGroupIds: [state.securityGroupId],
        ImageId: state.image.ImageId,
        InstanceType: state[provideInstanceType.name],
        // Availability Zones have capacity limitations that may impact your ability
 to launch instances.
        // The `RunInstances` operation will only succeed if it can allocate at
 least the `MinCount` of instances.
        // However, EC2 will attempt to launch up to the `MaxCount` of instances,
 even if the full request cannot be satisfied.
        // If you need a specific number of instances, use `MinCount` and `MaxCount`
 set to the same value.
        // If you want to launch up to a certain number of instances, use `MaxCount`
 and let EC2 provision as many as possible.
        // If you require a minimum number of instances, but do not want to exceed a
 maximum, use both `MinCount` and `MaxCount`.
        MinCount: 1,
        MaxCount: 1,
      }),
    );

    state.instanceId = Instances[0].InstanceId;

    try {
      // Poll `DescribeInstanceStatus` until status is "ok".
      await waitUntilInstanceStatusOk(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
```

```
        { InstanceIds: [Instances[0].InstanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logRunInstance = new ScenarioOutput(
  "logRunInstance",
  "The next step is to run your EC2 instance for the first time. This can take a few
 minutes.",
  { header: true, skipWhen: skipWhenErrors },
);

export const describeInstance = new ScenarioAction(
  "describeInstance",
  async (/** @type { State } */ state) => {
    /** @type { import("@aws-sdk/client-ec2").Instance[] } */
    const instances = [];

    try {
      const paginator = paginateDescribeInstances(
        {
          client: state.ec2Client,
        },
        {
          // Only get our created instance.
          InstanceIds: [state.instanceId],
        },
      );

      for await (const page of paginator) {
        for (const reservation of page.Reservations) {
          instances.push(...reservation.Instances);
        }
      }
      if (instances.length !== 1) {
```

```
        throw new Error(`Instance ${state.instanceId} not found.`);
      }

      // The only info we need is the IP address for SSH purposes.
      state.instanceIpAddress = instances[0].PublicIpAddress;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check provided values and try
 again.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSSHConnectionInfo = new ScenarioOutput(
  "logSSHConnectionInfo",
  (/** @type { State } */ state) =>
    `You can now SSH into your instance using the following command:
ssh -i ${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem ec2-user@
${state.instanceIpAddress}`,
  { preformatted: true, skipWhen: skipWhenErrors },
);

export const logStopInstance = new ScenarioOutput(
  "logStopInstance",
  "Stopping your EC2 instance.",
  { skipWhen: skipWhenErrors },
);

export const stopInstance = new ScenarioAction(
  "stopInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StopInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStopped(
        {
```

```
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  // Don't try to stop an instance that doesn't exist.
  { skipWhen: (/** @type { State } */ state) => !state.instanceId },
);

export const logIpAddressBehavior = new ScenarioOutput(
  "logIpAddressBehavior",
  [
    "When you run an instance, by default it's assigned an IP address.",
    "That IP address is not static. It will change every time the instance is
 restarted.",
    "The next step is to stop and restart your instance to demonstrate this
 behavior.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const logStartInstance = new ScenarioOutput(
  "logStartInstance",
  (/** @type { State } */ state) => `Starting instance ${state.instanceId}`,
  { skipWhen: skipWhenErrors },
);

export const startInstance = new ScenarioAction(
  "startInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StartInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
```

```
      );

      await waitUntilInstanceStatusOk(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logIpAllocation = new ScenarioOutput(
  "logIpAllocation",
  [
    "It is possible to have a static IP address.",
    "To demonstrate this, an IP will be allocated and associated to your EC2
 instance.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const allocateIp = new ScenarioAction(
  "allocateIp",
  async (/** @type { State } */ state) => {
    try {
      // An Elastic IP address is allocated to your AWS account, and is yours until
 you release it.
      const { AllocationId, PublicIp } = await state.ec2Client.send(
        new AllocateAddressCommand({}),
      );
      state.allocationId = AllocationId;
      state.allocatedIpAddress = PublicIp;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
```

```
        caught.message = `${caught.message}. Did you provide these values?`;
      }
      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const associateIp = new ScenarioAction(
  "associateIp",
  async (/** @type { State } */ state) => {
    try {
      // Associate an allocated IP address to an EC2 instance. An IP address can be
 allocated
      // with the AllocateAddress action.
      const { AssociationId } = await state.ec2Client.send(
        new AssociateAddressCommand({
          AllocationId: state.allocationId,
          InstanceId: state.instanceId,
        }),
      );
      state.associationId = AssociationId;
      // Update the IP address that is being tracked to match
      // the one just associated.
      state.instanceIpAddress = state.allocatedIpAddress;
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Did you provide the ID of a valid
 Elastic IP address AllocationId?`;
      }
      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logStaticIpProof = new ScenarioOutput(
  "logStaticIpProof",
  "The IP address should remain the same even after stopping and starting the
 instance.",
  { header: true, skipWhen: skipWhenErrors },
```

```
);

export const logCleanUp = new ScenarioOutput(
  "logCleanUp",
  "That's it! You can choose to clean up the resources now, or clean them up on your
 own later.",
  { header: true, skipWhen: skipWhenErrors },
);

export const confirmDisassociateAddress = new ScenarioInput(
  "confirmDisassociateAddress",
  "Do you want to disassociate and release the static IP address created earlier?",
  {
    type: "confirm",
    skipWhen: (/** @type { State } */ state) => !state.associationId,
  },
);

export const maybeDisassociateAddress = new ScenarioAction(
  "maybeDisassociateAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new DisassociateAddressCommand({
          AssociationId: state.associationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAssociationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid association
 ID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.associationId,
  },
);
```

```
export const maybeReleaseAddress = new ScenarioAction(
  "maybeReleaseAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new ReleaseAddressCommand({
          AllocationId: state.allocationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid AllocationID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.allocationId,
  },
);

export const confirmTerminateInstance = new ScenarioInput(
  "confirmTerminateInstance",
  "Do you want to terminate the instance?",
  // Don't do anything when an instance was never run.
  {
    skipWhen: (/** @type { State } */ state) => !state.instanceId,
    type: "confirm",
  },
);

export const maybeTerminateInstance = new ScenarioAction(
  "terminateInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new TerminateInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );
```

```
      await waitUntilInstanceTerminated(
        { client: state.ec2Client },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no instance to terminate or the
    // use chooses not to terminate.
    skipWhen: (/** @type { State } */ state) =>
      !state.instanceId || !state[confirmTerminateInstance.name],
  },
);

export const deleteTemporaryDirectory = new ScenarioAction(
  "deleteTemporaryDirectory",
  async (/** @type { State } */ state) => {
    try {
      await rm(state.tmpDirectory, { recursive: true });
    } catch (caught) {
      state.errors.push(caught);
    }
  },
);

export const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State}*/ state) => {
    const errorList = state.errors
      .map((err) => `• ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    preformatted: true,
    header: true,
    // Don't log errors when there aren't any!
```

```
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);
```

- API 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`

次のコード例は、`AllocateAddress` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { AllocateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Allocates an Elastic IP address to your AWS account.
 */
export const main = async () => {
  const client = new EC2Client({});
  const command = new AllocateAddressCommand({});

  try {
    const { AllocationId, PublicIp } = await client.send(command);
    console.log("A new IP address has been allocated to your account:");
    console.log(`ID: ${AllocationId} Public IP: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2.
 Look under Network & Security > Elastic IPs",
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
import { fileURLToPath } from "url";
// Call function if run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- API 詳細については、「 リファレンスAllocateAddress」の「」を参照してください。 AWS SDK for JavaScript API

`AssociateAddress`

次のコード例は、`AssociateAddress` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスAssociateAddress」の「」を参照してください。 AWS SDK for JavaScript API

`AuthorizeSecurityGroupIngress`

次の例は、`AuthorizeSecurityGroupIngress` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  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 詳細については、「 リファレンスAuthorizeSecurityGroupIngress」の「」を参照してください。 AWS SDK for JavaScript API

CreateKeyPair

次のコード例は、CreateKeyPair を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスCreateKeyPair」の「」を参照してください。 AWS SDK for JavaScript API

CreateLaunchTemplate

次の例は、CreateLaunchTemplate を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスCreateLaunchTemplate」の「」を参照してください。AWS SDK for JavaScript API

CreateSecurityGroup

次のコード例は、CreateSecurityGroup を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスCreateSecurityGroup」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteKeyPair`

次の例は、`DeleteKeyPair` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteKeyPair」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteLaunchTemplate`

次のコード例は、`DeleteLaunchTemplate` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- API 詳細については、「 リファレンスDeleteLaunchTemplate」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteSecurityGroup`

次の例は、`DeleteSecurityGroup` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteSecurityGroup」の「」を参照してください。AWS SDK for JavaScript API

`DescribeAddresses`

次のコード例は、`DescribeAddresses` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeAddresses」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeIamInstanceProfileAssociations`

次の例は、`DescribeIamInstanceProfileAssociations` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
    const ec2Client = new EC2Client({});
    const { IamInstanceProfileAssociations } = await ec2Client.send(
```

```
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
```

- API 詳細については、「 リファレンスDescribeIamInstanceProfileAssociations」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeImages`

次のコード例は、`DescribeImages` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { EC2Client, paginateDescribeImages } from "@aws-sdk/client-ec2";

/**
 * Describes the specified images (AMIs, AKIs, and ARIs) available to you or all of
 the images available to you.
 * @param {{ architecture: string, pageSize: number }} options
 */
export const main = async ({ architecture, pageSize }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});

  // The paginate function is a wrapper around the base command.
  const paginator = paginateDescribeImages(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the base command.
    { client, pageSize },
    {
```

```
      // There are almost 70,000 images available. Be specific with your filtering
      // to increase efficiency.
      // See https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
ec2/interfaces/describeimagescommandinput.html#filters
      Filters: [{ Name: "architecture", Values: [architecture] }],
    },
  );

  /**
   * @type {import('@aws-sdk/client-ec2').Image[]}
   */
  const images = [];
  let recordsScanned = 0;

  try {
    for await (const page of paginator) {
      recordsScanned += pageSize;
      if (page.Images.length) {
        images.push(...page.Images);
        break;
      } else {
        console.log(
          `No matching image found yet. Searched ${recordsScanned} records.`,
        );
      }
    }

    if (images.length) {
      console.log(
        `Found ${images.length} images:\n\n${images.map((image) =>
 image.Name).join("\n")}\n`,
      );
    } else {
      console.log(
        `No matching images found. Searched ${recordsScanned} records.\n`,
      );
    }

    return images;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
      return [];
    }
```

```
    throw caught;
  }
};
```

- API 詳細については、「 リファレンスDescribeImages」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeInstanceTypes`

次のコード例は、DescribeInstanceTypes を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { EC2Client, paginateDescribeInstanceTypes } from "@aws-sdk/client-ec2";

/**
 * Describes the specified instance types. By default, all instance types for the
 * current Region are described. Alternatively, you can filter the results.
 * @param {{ pageSize: string, supportedArch: string[], freeTier: boolean }} options
 */
export const main = async ({ pageSize, supportedArch, freeTier }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});

  // The paginate function is a wrapper around the underlying command.
  const paginator = paginateDescribeInstanceTypes(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the underlying command.
    { client, pageSize },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
```

```
          Values: supportedArch,
        },
        { Name: "free-tier-eligible", Values: [freeTier ? "true" : "false"] },
      ],
    },
  );

  try {
    /**
     * @type {import('@aws-sdk/client-ec2').InstanceTypeInfo[]}
     */
    const instanceTypes = [];

    for await (const page of paginator) {
      if (page.InstanceTypes.length) {
        instanceTypes.push(...page.InstanceTypes);

        // When we have at least 1 result, we can stop.
        if (instanceTypes.length >= 1) {
          break;
        }
      }
    }
    console.log(
      `Memory size in MiB for matching instance types:\n\n${instanceTypes.map((it)
 => `${it.InstanceType}: ${it.MemoryInfo.SizeInMiB} MiB`).join("\n")}`,
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
      return [];
    }
    throw caught;
  }
};
```

- API 詳細については、「 リファレンスDescribeInstanceTypes」の「」を参照してください。AWS SDK for JavaScript API

`DescribeInstances`

次のコード例は、DescribeInstances を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { EC2Client, paginateDescribeInstances } from "@aws-sdk/client-ec2";

/**
 * List all of your EC2 instances running with the provided architecture that
 * were launched in the past month.
 * @param {{ pageSize: string, architectures: string[] }} options
 */
export const main = async ({ pageSize, architectures }) => {
  pageSize = parseInt(pageSize);
  const client = new EC2Client({});
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;

  const paginator = paginateDescribeInstances(
    {
      client,
      pageSize,
    },
    {
      Filters: [
        { Name: "architecture", Values: architectures },
        { Name: "instance-state-name", Values: ["running"] },
        {
          Name: "launch-time",
          Values: [launchTimePattern],
        },
      ],
    },
  );

  try {
    /**
```

```
     * @type {import('@aws-sdk/client-ec2').Instance[]}
     */
    const instanceList = [];
    for await (const page of paginator) {
      const { Reservations } = page;
      Reservations.forEach((r) => instanceList.push(...r.Instances));
    }
    console.log(
      `Running instances launched this month:\n\n${instanceList.map((instance) =>
 instance.InstanceId).join("\n")}`,
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- API 詳細については、「 リファレンスDescribeInstances」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeKeyPairs`

次の例は、`DescribeKeyPairs` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeKeyPairs」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeRegions`

次の例は、`DescribeRegions` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeRegions」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeSecurityGroups`

次の例は、`DescribeSecurityGroups` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeSecurityGroups」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeSubnets`

次のコード例は、`DescribeSubnets` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスDescribeSubnets」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeVpcs`

次のコード例は、`DescribeVpcs` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
```

- API 詳細については、「 リファレンスDescribeVpcs」の「」を参照してください。 AWS SDK for JavaScript API

`DisassociateAddress`

次の例は、`DisassociateAddress` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDisassociateAddress」の「」を参照してください。AWS SDK for JavaScript API

`MonitorInstances`

次のコード例は、MonitorInstances を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスMonitorInstances」の「」を参照してください。 AWS SDK for JavaScript API

`RebootInstances`

次の例は、RebootInstances を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスRebootInstances」の「」を参照してください。 AWS SDK for JavaScript API

`ReleaseAddress`

次の例は、`ReleaseAddress` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスReleaseAddress」の「」を参照してください。 AWS SDK for JavaScript API

`ReplaceIamInstanceProfileAssociation`

次のコード例は、`ReplaceIamInstanceProfileAssociation` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );
```

- API 詳細については、「 リファレンスReplaceIamInstanceProfileAssociation」の「」を参照してください。 AWS SDK for JavaScript API

`RunInstances`

次の例は、`RunInstances` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { EC2Client, RunInstancesCommand } from "@aws-sdk/client-ec2";

/**
```

```
 * Create new EC2 instances.
 * @param {{
 *  keyName: string,
 *  securityGroupIds: string[],
 *  imageId: string,
 *  instanceType: import('@aws-sdk/client-ec2')._InstanceType,
 *  minCount?: number,
 *  maxCount?: number }} options
 */
export const main = async ({
  keyName,
  securityGroupIds,
  imageId,
  instanceType,
  minCount = "1",
  maxCount = "1",
}) => {
  const client = new EC2Client({});
  minCount = parseInt(minCount);
  maxCount = parseInt(maxCount);
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: keyName,
    // Your security group.
    SecurityGroupIds: securityGroupIds,
    // An Amazon Machine Image (AMI). There are multiple ways to search for AMIs.
 For more information, see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html
    ImageId: imageId,
    // An instance type describing the resources provided to your instance. There
 are multiple
    // ways to search for instance types. For more information see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-discovery.html
    InstanceType: instanceType,
    // Availability Zones have capacity limitations that may impact your ability to
 launch instances.
    // The `RunInstances` operation will only succeed if it can allocate at least
 the `MinCount` of instances.
    // However, EC2 will attempt to launch up to the `MaxCount` of instances, even
 if the full request cannot be satisfied.
    // If you need a specific number of instances, use `MinCount` and `MaxCount` set
 to the same value.
    // If you want to launch up to a certain number of instances, use `MaxCount` and
 let EC2 provision as many as possible.
```

```
    // If you require a minimum number of instances, but do not want to exceed a
 maximum, use both `MinCount` and `MaxCount`.
    MinCount: minCount,
    MaxCount: maxCount,
  });

  try {
    const { Instances } = await client.send(command);
    const instanceList = Instances.map(
      (instance) => `• ${instance.InstanceId}`,
    ).join("\n");
    console.log(`Launched instances:\n${instanceList}`);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceCountExceeded") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- API 詳細については、「 リファレンスRunInstances」の「」を参照してください。 AWS SDK for JavaScript API

`StartInstances`

次のコード例は、StartInstances を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { EC2Client, StartInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";
```

```
/**
 * Starts an Amazon EBS-backed instance that you've previously stopped.
 * @param {{ instanceIds }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StartInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- API 詳細については、「 リファレンスStartInstances」の「」を参照してください。 AWS SDK for JavaScript API

`StopInstances`

次の例は、StopInstances を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { EC2Client, StopInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Stop one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StopInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- API 詳細については、「 リファレンスStopInstances」の「」を参照してください。 AWS SDK for JavaScript API

`TerminateInstances`

次のコード例は、`TerminateInstances` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { EC2Client, TerminateInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Terminate one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new TerminateInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
```

```
    } else {
      throw caught;
    }
  }
  ``;
};
```

- API 詳細については、「 リファレンスTerminateInstances」の「」を参照してください。 AWS SDK for JavaScript API

`UnmonitorInstances`

次の例は、UnmonitorInstances を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { EC2Client, UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "url";
import { parseArgs } from "util";

/**
 * Turn off detailed monitoring for the selected instance.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new UnmonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instanceMonitoringsList = InstanceMonitorings.map(
      (im) =>
```

```
        `• Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instanceMonitoringsList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- API 詳細については、「 リファレンスUnmonitorInstances」の「」を参照してください。 AWS SDK for JavaScript API

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、インスタンスの数を指定された範囲内に保持します。
- Elastic Load Balancing を使用してHTTPリクエストを処理し、配信します。 Elastic Load Balancing
- 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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 "url";

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

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),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
```

```
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
```

```
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

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(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
```

```
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
```

```
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
```

```
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
```

```
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
```

```
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
```

```
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
```

```
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
```

```
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
```

```
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
```

```
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
```

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

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
```

```
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

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

- API 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 - JavaScript (v3) SDK用の 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- API 詳細については、「 リファレンスDescribeLoadBalancers」の「」を参照してください。AWS SDK for JavaScript API

トピック

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスCreateListener」の「」を参照してください。 AWS SDK for JavaScript API

`CreateLoadBalancer`

次のコード例は、`CreateLoadBalancer` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスCreateLoadBalancer」の「」を参照してください。AWS SDK for JavaScript API

`CreateTargetGroup`

次の例は、`CreateTargetGroup` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスCreateTargetGroup」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteLoadBalancer`

次のコード例は、`DeleteLoadBalancer` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスDeleteLoadBalancer」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteTargetGroup`

次のコード例は、`DeleteTargetGroup` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスDeleteTargetGroup」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeLoadBalancers`

次の例は、`DescribeLoadBalancers` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- API 詳細については、「 リファレンスDescribeLoadBalancers」の「」を参照してください。AWS SDK for JavaScript API

`DescribeTargetGroups`

次のコード例は、`DescribeTargetGroups` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

- API 詳細については、「 リファレンスDescribeTargetGroups」の「」を参照してください。AWS SDK for JavaScript API

`DescribeTargetHealth`

次のコード例は、`DescribeTargetHealth` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

- API 詳細については、「 リファレンスDescribeTargetHealth」の「」を参照してください。AWS SDK for JavaScript API

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、インスタンスの数を指定された範囲内に保持します。
- Elastic Load Balancing を使用してHTTPリクエストを処理し、配信します。 Elastic Load Balancing
- 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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 "url";

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

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),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
```

```
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

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(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
```

```
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
```

```
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
```

```
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
```

```
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
```

```
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
```

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

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
```

```
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
```

```
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
```

```
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

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

```
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
```

```
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

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

- API 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

EventBridge JavaScript (v3) SDK に を使用する 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 EventBridge。

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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import {
  EventBridgeClient,
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";

export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutEventsCommand({
      Entries: [
        {
          Detail: JSON.stringify({ greeting: "Hello there." }),
          DetailType: detailType,
```

```
          Resources: resources,
          Source: source,
        },
      ],
    }),
  );

  console.log("PutEvents response:");
  console.log(response);
  // PutEvents response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- API 詳細については、「 リファレンスPutEvents」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

ebevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- API 詳細については、「 リファレンスPutEvents」の「」を参照してください。 AWS SDK for JavaScript API

`PutRule`

次の例は、PutRule を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";
```

```
export const putRule = async (
  ruleName = "some-rule",
  source = "some-source",
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutRuleCommand({
      Name: ruleName,
      EventPattern: JSON.stringify({ source: [source] }),
      State: "ENABLED",
      EventBusName: "default",
    }),
  );

  console.log("PutRule response:");
  console.log(response);
  // PutRule response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/
EventBridgeTestRule-1696280037720'
  // }
  return response;
};
```

- API 詳細については、「 リファレンスPutRule」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

ebevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- API 詳細については、「 リファレンスPutRule」の「」を参照してください。 AWS SDK for JavaScript API

`PutTargets`

次の例は、PutTargets を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
import {
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   FailedEntries: [],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- API 詳細については、「 リファレンスPutTargets」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myEventBridgeTarget",
    },
  ],
};

ebevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- API 詳細については、「 リファレンスPutTargets」の「」を参照してください。 AWS SDK for JavaScript API

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 の JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールイベントを作成する方法を示します。cron 式 EventBridge 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 を設定します。この例では、Lambda JavaScript ランタイム を使用して Lambda 関数を作成しますAPI。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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

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
- EventBridge
- Lambda
- Amazon SNS

AWS Glue JavaScript (v3) SDK に を使用する 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスListJobs」の「」を参照してください。 AWS SDK for JavaScript API

トピック

- 基礎
- アクション

基礎

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します。
- 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データを抽出し、データを変換して、JSONフォーマットされた出力を別の S3 バケットにロードするジョブを作成します。
- 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

パブリック Amazon Simple Storage Service (Amazon S3) バケットをクロールし、見つかった CSV形式の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データを抽出し、フィールドを削除して名前を変更して変換し、 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":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "RUNNING":
      break;
    case "SUCCEEDED":
      return;
    default:
      throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
  }
```

```
  log(
    `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
  );
  await wait(waitTimeInSeconds);
  return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
 view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);
```

```
    return { ...context };
  };
```

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 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
  - GetJobRun
  - 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスCreateCrawler」の「」を参照してください。 AWS SDK for JavaScript API

`CreateJob`

次の例は、`CreateJob` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスCreateJob」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteCrawler`

次の例は、`DeleteCrawler` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

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

- API 詳細については、「 リファレンスDeleteCrawler」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteDatabase`

次の例は、DeleteDatabase を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

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

- API 詳細については、「 リファレンスDeleteDatabase」の「」を参照してください。 AWS SDK for JavaScript API

DeleteJob

次のコード例は、DeleteJob を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

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

- API 詳細については、「 リファレンスDeleteJob」の「」を参照してください。 AWS SDK for JavaScript API

DeleteTable

次のコード例は、DeleteTable を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const deleteTable = (databaseName, tableName) => {
```

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

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

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

- API 詳細については、「 リファレンスDeleteTable」の「」を参照してください。 AWS SDK for JavaScript API

`GetCrawler`

次の例は、`GetCrawler` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

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

- API 詳細については、「 リファレンスGetCrawler」の「」を参照してください。 AWS SDK for JavaScript API

GetDatabase

次のコード例は、GetDatabase を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

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

- API 詳細については、「 リファレンスGetDatabase」の「」を参照してください。 AWS SDK for JavaScript API

GetDatabases

次のコード例は、GetDatabases を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const getDatabases = () => {
```

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

  const command = new GetDatabasesCommand({});

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

- API 詳細については、「 リファレンスGetDatabases」の「」を参照してください。 AWS SDK for JavaScript API

GetJob

次のコード例は、GetJob を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

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

- API 詳細については、「 リファレンスGetJob」の「」を参照してください。 AWS SDK for JavaScript API

GetJobRun

次のコード例は、GetJobRun を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

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

- API 詳細については、「 リファレンスGetJobRun」の「」を参照してください。 AWS SDK for JavaScript API

`GetJobRuns`

次の例は、`GetJobRuns` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });
```

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

- API 詳細については、「 リファレンスGetJobRuns」の「」を参照してください。 AWS SDK for JavaScript API

`GetTables`

次のコード例は、`GetTables` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

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

- API 詳細については、「 リファレンスGetTables」の「」を参照してください。 AWS SDK for JavaScript API

`ListJobs`

次のコード例は、`ListJobs` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

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

- API 詳細については、「 リファレンスListJobs」の「」を参照してください。 AWS SDK for JavaScript API

`StartCrawler`

次の例は、StartCrawler を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });
```

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

- API 詳細については、「 リファレンスStartCrawler」の「」を参照してください。 AWS SDK for JavaScript API

`StartJobRun`

次の例は、StartJobRun を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスStartJobRun」の「」を参照してください。 AWS SDK for JavaScript API

HealthImaging JavaScript (v3) SDK に を使用する 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 HealthImaging。

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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

開始方法

こんにち HealthImagingは

次のコード例は、 の使用を開始する方法を示しています HealthImaging。

SDK の 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 詳細については、「 リファレンスListDatastores」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 の 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 = "xxxxxxxxxxx",
    imageSetId = "xxxxxxxxxxxx",
    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:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
        //             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:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
        //             imageSetId: 'xxxxxxxxxxxxxxx',
        //             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(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "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(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
    "12345678901234567890123456789012",
    "1",
    false,
);
```

イメージセットのサブセットを送信先にコピーし、強制的にコピーします。

```
await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
    "12345678901234567890123456789012",
    "1",
    true,
    ["12345678901234567890123456789012", "11223344556677889900112233445566"]
);
```

- API 詳細については、「 リファレンスCopyImageSet」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`CreateDatastore`

次のコード例は、`CreateDatastore` を使用する方法を示しています。

SDK の 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 詳細については、「 リファレンスCreateDatastore」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`DeleteDatastore`

次のコード例は、`DeleteDatastore` を使用する方法を示しています。

SDK の 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 詳細については、「 リファレンスDeleteDatastore」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`DeleteImageSet`

次の例は、DeleteImageSet を使用する方法を説明しています。

SDK の 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 詳細については、「 リファレンスDeleteImageSet」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`GetDICOMImportJob`

次の例は、`GetDICOMImportJob` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
```

```
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- API 詳細については、「 リファレンス」の「G etDICOMImportジョブ」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`GetDatastore`

次のコード例は、`GetDatastore` を使用する方法を示しています。

SDK JavaScript (v3) 用の

```
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 詳細については、「 リファレンスGetDatastore」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`GetImageFrame`

次のコード例は、`GetImageFrame` を使用する方法を示しています。

SDK JavaScript (v3) 用の

```
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 詳細については、「 リファレンスGetImageFrame」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`GetImageSet`

次のコード例は、`GetImageSet` を使用する方法を示しています。

SDK JavaScript (v3) 用の

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
```

```
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- API 詳細については、「 リファレンスGetImageSet」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`GetImageSetMetadata`

次のコード例は、`GetImageSetMetadata` を使用する方法を示しています。

SDK JavaScript (v3) 用の

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 "fs";
```

```
/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

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 詳細については、「 リファレンスGetImageSetMetadata」の「」を参照してください。AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`ListDICOMImportJobs`

次の例は、`ListDICOMImportJobs` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
```

```
  // }
  // ]}

  return jobSummaries;
};
```

- API 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「L istDICOMImport ジョブ」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`ListDatastores`

次のコード例は、`ListDatastores` を使用する方法を示しています。

SDK JavaScript (v3) 用の

```
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 詳細については、「 リファレンスListDatastores」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`ListImageSetVersions`

次のコード例は、`ListImageSetVersions` を使用する方法を示しています。

SDK JavaScript (v3) 用の

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
```

```
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- API 詳細については、「 リファレンスListImageSetVersions」の「」を参照してください。AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`ListTagsForResource`

次のコード例は、`ListTagsForResource` を使用する方法を示しています。

SDK JavaScript (v3) 用の

```
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 詳細については、「 リファレンスListTagsForResource」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`SearchImageSets`

次の例は、SearchImageSets を使用する方法を説明しています。

SDK JavaScript (v3) 用の

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および を使用するBETWEEN演算子DICOMStudyTime。

```
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: を使用するBETWEEN演算子createdAt。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: EQUAL演算子 on DICOMSeriesInstanceUIDと BETWEEN on updatedAt および は、レスポンスを updatedAt フィールドのASC順にソートします。

```
    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 詳細については、「 リファレンスSearchImageSets」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`StartDICOMImportJob`

次のコード例は、StartDICOMImportJob を使用する方法を示しています。

SDK JavaScript (v3) 用の

```
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 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「S tartDICOMImport ジョブ」を参照してください。

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`TagResource`

次の例は、`TagResource` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

```
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 詳細については、「 リファレンスTagResource」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`UntagResource`

次の例は、UntagResource を使用する方法を説明しています。

SDK JavaScript (v3) 用の

```
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 詳細については、「 リファレンスUntagResource」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`UpdateImageSetMetadata`

次のコード例は、`UpdateImageSetMetadata` を使用する方法を示しています。

SDK JavaScript (v3) 用の

```
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す。

```
    const updateMetadata = {
        "revertToVersionId": "1"
    };

    await updateImageSetMetadata(datastoreID, imageSetID,
        versionID, updateMetadata);
```

- API 詳細については、「 リファレンスUpdateImageSetMetadata」の「」を参照してください。 AWS SDK for JavaScript API

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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、 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 HealthImaging。

実装は、ワークフローの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 インポート用のリソースを設定します。
- データストアにDICOMファイルをインポートします。
- インポートジョブIDsのイメージセットを取得します。
- 画像セットIDsの画像フレームを取得します。
- 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 JavaScript (v3) 用の

index.js - ステップをオーケストレーションします。

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

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
```

```
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
```

```
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

deploy-steps.js - リソースをデプロイします。

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
```

```
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
```

```
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
```

```
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName == state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

dataset-steps.js - DICOM ファイルをコピーします。

```
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
```

```
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;
```

```
      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js - データストアへのインポートを開始します。

```
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */
```

```
export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
```

```
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

image-set-steps.js - 画像セット を取得しますIDs。

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

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

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
```

```
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

image-frame-steps.js - イメージフレーム を取得しますIDs。

```
import {
```

```
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

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

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
```

```
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
```

```
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
  { slow: false },
);
```

verify-steps.js - イメージフレームを確認します。AWS HealthImaging Pixel Data Verification 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}`,
                  ),
                );
              }
            });
          });
        }
      }
    }
  },
);
```

clean-up-steps.js - リソースを破棄します。

```
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 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
  - G etDICOMImportジョブ
  - GetImageFrame
  - GetImageSetMetadata

- SearchImageSets
- S tartDICOMImportジョブ

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 JavaScript (v3) 用の

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 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**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

イメージセットにタグを付ける

次のコード例は、 HealthImaging 画像セットにタグを付ける方法を示しています。

SDK JavaScript (v3) 用の

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 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**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

IAM JavaScript (v3) SDK に を使用する 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IAM。

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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

開始方法

Hello IAM

次のコード例は、 の使用を開始する方法を示していますIAM。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
 async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
```

```
  for await (const page of paginator) {
    if (page.Policies) {
      page.Policies.forEach((p) => {
        console.log(`${p.PolicyName}`);
        policyCount++;
      });
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- API 詳細については、「 リファレンスListPolicies」の「」を参照してください。 AWS SDK for JavaScript API

トピック

- アクション
- シナリオ

アクション

`AttachRolePolicy`

次のコード例は、AttachRolePolicy を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスAttachRolePolicy」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        console.log(
          "AmazonDynamoDBFullAccess is already attached to this role."
        );
        process.exit();
      }
    });
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.attachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

- 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、「 リファレンスAttachRolePolicy」の「」を参照してください。 AWS SDK for JavaScript API

`CreateAccessKey`

次のコード例は、`CreateAccessKey` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスCreateAccessKey」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- 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、「 リファレンスCreateAccessKey」の「」を参照してください。 AWS SDK for JavaScript API

`CreateAccountAlias`

次の例は、`CreateAccountAlias` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスCreateAccountAlias」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスCreateAccountAlias」の「」を参照してください。 AWS SDK for JavaScript API

`CreateGroup`

次のコード例は、`CreateGroup` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスCreateGroup」の「」を参照してください。 AWS SDK for JavaScript API

`CreateInstanceProfile`

次のコード例は、`CreateInstanceProfile` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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

- API 詳細については、「 リファレンスCreateInstanceProfile」の「」を参照してください。 AWS SDK for JavaScript API

`CreatePolicy`

次のコード例は、CreatePolicy を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスCreatePolicy」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
```

```
    {
      Effect: "Allow",
      Action: [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
      ],
      Resource: "RESOURCE_ARN",
    },
  ],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

iam.createPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスCreatePolicy」の「」を参照してください。 AWS SDK for JavaScript API

CreateRole

次の例は、CreateRole を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスCreateRole」の「」を参照してください。 AWS SDK for JavaScript API

`CreateSAMLProvider`

次の例は、`CreateSAMLProvider` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import * as path from "path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";

const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.
 * For more information on generating this document,
 see https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_providers_create_saml.html#samlstep1.
 */
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });
```

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

- API 詳細については、「 リファレンス」の「CreateSAMLProvider」を参照してください。 AWS SDK for JavaScript API

`CreateServiceLinkedRole`

次の例は、CreateServiceLinkedRole を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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" }),
      );
    } else {
      throw caught;
    }
  }
};
```

- API 詳細については、「 リファレンスCreateServiceLinkedRole」の「」を参照してください。AWS SDK for JavaScript API

`CreateUser`

次のコード例は、`CreateUser` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスCreateUser」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
```

```
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

- 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、「 リファレンスCreateUser」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteAccessKey`

次のコード例は、DeleteAccessKey を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteAccessKey」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

iam.deleteAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
```

```
  }
});
```

- 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、「 リファレンスDeleteAccessKey」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteAccountAlias`

次のコード例は、DeleteAccountAlias を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteAccountAlias」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスDeleteAccountAlias」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteGroup`

次の例は、`DeleteGroup` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteGroup」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteInstanceProfile`

次の例は、`DeleteInstanceProfile` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
```

- API 詳細については、「 リファレンスDeleteInstanceProfile」の「」を参照してください。 AWS SDK for JavaScript API

`DeletePolicy`

次の例は、`DeletePolicy` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeletePolicy」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteRole`

次の例は、`DeleteRole` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteRole」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteRolePolicy`

次のコード例は、`DeleteRolePolicy` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteRolePolicy」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteSAMLProvider`

次のコード例は、DeleteSAMLProvider を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンス」の「DeleteSAMLProvider」を参照してください。 AWS SDK for JavaScript API

`DeleteServerCertificate`

次の例は、DeleteServerCertificate を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteServerCertificate」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 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、「 リファレンスDeleteServerCertificate」の「」を参照してください。 AWS SDK for JavaScript API

DeleteServiceLinkedRole

次の例は、DeleteServiceLinkedRole を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteServiceLinkedRole」の「」を参照してください。 AWS SDK for JavaScript API

DeleteUser

次の例は、DeleteUser を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteUser」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
```

```
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- 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、「 リファレンスDeleteUser」の「」を参照してください。 AWS SDK for JavaScript API

`DetachRolePolicy`

次の例は、DetachRolePolicy を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDetachRolePolicy」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        var params = {
          PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
```

```
          RoleName: process.argv[2],
        };
        iam.detachRolePolicy(params, function (err, data) {
          if (err) {
            console.log("Unable to detach policy from role", err);
          } else {
            console.log("Policy detached from role successfully");
            process.exit();
          }
        });
      }
    });
  }
});
```

- 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、「 リファレンスDetachRolePolicy」の「」を参照してください。 AWS SDK for JavaScript API

`GetAccessKeyLastUsed`

次の例は、GetAccessKeyLastUsed を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスGetAccessKeyLastUsed」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getAccessKeyLastUsed(
  { AccessKeyId: "ACCESS_KEY_ID" },
```

```
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
    }
  }
);
```

- 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、「 リファレンスGetAccessKeyLastUsed」の「」を参照してください。 AWS SDK for JavaScript API

`GetAccountPasswordPolicy`

次のコード例は、`GetAccountPasswordPolicy` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスGetAccountPasswordPolicy」の「」を参照してください。 AWS SDK for JavaScript API

`GetPolicy`

次の例は、`GetPolicy` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスGetPolicy」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

- 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、「 リファレンスGetPolicy」の「」を参照してください。 AWS SDK for JavaScript API

`GetRole`

次の例は、`GetRole` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスGetRole」の「」を参照してください。 AWS SDK for JavaScript API

`GetServerCertificate`

次の例は、`GetServerCertificate` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスGetServerCertificate」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });
```

```
iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 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、「 リファレンスGetServerCertificate」の「」を参照してください。 AWS SDK for JavaScript API

`GetServiceLinkedRoleDeletionStatus`

次の例は、`GetServiceLinkedRoleDeletionStatus` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  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 詳細については、「 リファレンスGetServiceLinkedRoleDeletionStatus」の「」を参照してください。 AWS SDK for JavaScript API

`ListAccessKeys`

次の例は、ListAccessKeys を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListAccessKeys」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};

iam.listAccessKeys(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスListAccessKeys」の「」を参照してください。 AWS SDK for JavaScript API

`ListAccountAliases`

次の例は、ListAccountAliases を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListAccountAliases」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスListAccountAliases」の「」を参照してください。 AWS SDK for JavaScript API

`ListAttachedRolePolicies`

次のコード例は、`ListAttachedRolePolicies` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListAttachedRolePolicies」の「」を参照してください。AWS SDK for JavaScript API

ListGroups

次のコード例は、ListGroups を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListGroups」の「」を参照してください。 AWS SDK for JavaScript API

`ListPolicies`

次のコード例は、`ListPolicies` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListPolicies」の「」を参照してください。 AWS SDK for JavaScript API

`ListRolePolicies`

次のコード例は、`ListRolePolicies` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListRolePolicies」の「」を参照してください。 AWS SDK for JavaScript API

`ListRoles`

次のコード例は、ListRoles を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListRoles」の「」を参照してください。 AWS SDK for JavaScript API

`ListSAMLProviders`

次の例は、`ListSAMLProviders` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンス」の「ListSAMLProviders」を参照してください。 AWS SDK for JavaScript API

`ListServerCertificates`

次の例は、`ListServerCertificates` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListServerCertificates」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスListServerCertificates」の「」を参照してください。 AWS SDK for JavaScript API

`ListUsers`

次の例は、`ListUsers` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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);
  response.Users?.forEach(({ UserName, CreateDate }) => {
    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 詳細については、「 リファレンスListUsers」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
      console.log("User " + user.UserName + " created", user.CreateDate);
    });
  }
});
```

- 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、「 リファレンスListUsers」の「」を参照してください。 AWS SDK for JavaScript API

`PutRolePolicy`

次のコード例は、`PutRolePolicy` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
```

```
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::some-test-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

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

```
  return response;
};
```

- API 詳細については、「 リファレンスPutRolePolicy」の「」を参照してください。 AWS SDK for JavaScript API

`UpdateAccessKey`

次のコード例は、UpdateAccessKey を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスUpdateAccessKey」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};

iam.updateAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスUpdateAccessKey」の「」を参照してください。 AWS SDK for JavaScript API

`UpdateServerCertificate`

次のコード例は、`UpdateServerCertificate` を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスUpdateServerCertificate」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

iam.updateServerCertificate(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスUpdateServerCertificate」の「」を参照してください。 AWS SDK for JavaScript API

`UpdateUser`

次のコード例は、UpdateUser を使用する方法を示しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスUpdateUser」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

iam.updateUser(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスUpdateUser」の「」を参照してください。 AWS SDK for JavaScript API

`UploadServerCertificate`

次の例は、`UploadServerCertificate` を使用する方法を説明しています。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
```

```
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
 the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }

    throw err;
  }
};

/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });
```

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

- API 詳細については、「 リファレンスUploadServerCertificate」の「」を参照してください。 AWS SDK for JavaScript API

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、インスタンスの数を指定された範囲内に保持します。
- Elastic Load Balancing を使用してHTTPリクエストを処理し、配信します。 Elastic Load Balancing
- 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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 "url";

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

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),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
```

```
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
```

```
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

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(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
```

```
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
```

```
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
```

```
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
```

```
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
```

```
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
```

```
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
```

```
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
```

```
    return MESSAGES.detachedSsmOnlyCustomRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  }
}),
new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
      }),
    );
  } catch (e) {
    state.detachSsmOnlyAWSRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
  if (state.detachSsmOnlyAWSRolePolicyError) {
    console.error(state.detachSsmOnlyAWSRolePolicyError);
    return MESSAGES.detachSsmOnlyAWSRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  } else {
    return MESSAGES.detachedSsmOnlyAWSRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  }
}),
new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
  if (state.deleteSsmOnlyInstanceProfileError) {
```

```
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
```

```
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
```

```
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
```

```
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

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

- API 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

ユーザーを作成してロールを引き受ける

シナリオ
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**
>
> セキュリティ上のリスクを回避するため、専用ソフトウェアの開発時や実際のデータの使用時には、認証にIAMユーザーを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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon S3 バケットを一覧表示するアクセス許可を付与する IAM ユーザーと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 = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
```

```
      return User;
    } else {
      throw caught;
    }
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
 STS).
  // It's not best practice to use access keys. For more information, see https://
aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;

  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
```

```
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
      return await listBuckets(s3Client);
    } catch (err) {
      if (err instanceof Error && err.name === "InvalidAccessKeyId") {
        throw err;
      }
    }
  });

  // Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
  // is thrown while the user and access keys are still stabilizing.
  const { Role } = await retry(
    {
      intervalInMs: 2000,
      maxRetries: 60,
    },
    () =>
      iamClient.send(
        new CreateRoleCommand({
          AssumeRolePolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
              {
                Effect: "Allow",
                Principal: {
                  // Allow the previously created user to assume this role.
                  AWS: User.Arn,
                },
                Action: "sts:AssumeRole",
              },
            ],
          }),
          RoleName: roleName,
        }),
      ),
  );

  if (!Role) {
    throw new Error("Role not created");
  }

  // Create a policy that allows the user to list S3 buckets.
```

```
  const { Policy: listBucketPolicy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Action: ["s3:ListAllMyBuckets"],
            Resource: "*",
          },
        ],
      }),
      PolicyName: policyName,
    }),
  );

  if (!listBucketPolicy) {
    throw new Error("Policy not created");
  }

  // Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
  await iamClient.send(
    new AttachRolePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
      RoleName: Role.RoleName,
    }),
  );

  // Assume the role.
  const stsClient = new STSClient({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the assume role operation until it succeeds.
  const { Credentials } = await retry(
    { intervalInMs: 2000, maxRetries: 60 },
    () =>
      stsClient.send(
        new AssumeRoleCommand({
          RoleArn: Role.Arn,
          RoleSessionName: `iamBasicScenarioSession-${Math.floor(
```

```
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

await iamClient.send(
  new DeletePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
  }),
);

await iamClient.send(
  new DeleteRoleCommand({
    RoleName: Role.RoleName,
  }),
```

```
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- API 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

for JavaScript (v3) を使用した SDK 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。

各例には、完全なソースコードへのリンクが含まれています。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック
- サーバーレスサンプル

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ログ記録します。

SDK JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で Kinesis イベントを消費する JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

を使用して Lambda で Kinesis イベントを消費する TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
```

```
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

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 JavaScript (v3) 用の

> **Note**
>
> については、「」を参照してください GitHub。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;
}
```

を使用して Lambda で Kinesis バッチアイテムの失敗をレポートする TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
```

```
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

for JavaScript (v3) を使用する Lambda SDK の例

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。

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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスListFunctions」の「」を参照してください。 AWS SDK for JavaScript API

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

`CreateFunction`

次のコード例は、CreateFunction を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスCreateFunction」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteFunction`

次の例は、`DeleteFunction` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- API 詳細については、「 リファレンスDeleteFunction」の「」を参照してください。 AWS SDK for JavaScript API

`GetFunction`

次の例は、`GetFunction` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- API 詳細については、「 リファレンスGetFunction」の「」を参照してください。 AWS SDK for JavaScript API

`Invoke`

次の例は、Invoke を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「呼び出し」を参照してください。

`ListFunctions`

次のコード例は、`ListFunctions` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

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

- API 詳細については、「 リファレンスListFunctions」の「」を参照してください。 AWS SDK for JavaScript API

`UpdateFunctionCode`

次の例は、`UpdateFunctionCode` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
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 詳細については、「 リファレンスUpdateFunctionCode」の「」を参照してください。AWS SDK for JavaScript API

`UpdateFunctionConfiguration`

次の例は、`UpdateFunctionConfiguration` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
const updateFunctionConfiguration = (funcName) => {
```

```
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

- API 詳細については、「 リファレンスUpdateFunctionConfiguration」の「」を参照してください。 AWS SDK for JavaScript API

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 の 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セットアップと実行の手順については、「」の詳細な例を参照してください GitHub。

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 ゲートウェイ
- 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 の 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、「」の「 GitHub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

関数の使用を開始します

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

ログに書き込むためのアクセス許可を Lambda に付与する AWS Identity and Access Management （ IAM) ロールを作成します。

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

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 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

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 JavaScript (v2) 用の

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。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Lambda

SDK の 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 AWS SDK for JavaScript v3 を使用します。

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、「」の詳細な例を参照してくださいGitHub。

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 の JavaScript (v3)

Lambda JavaScript ランタイム を使用して AWS Lambda 関数を作成する方法を示しますAPI。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作業記念日を確認し、Amazon Simple Notification Service (Amazon SNS) を使用して、1 年間の記念日に従業員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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

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 ゲートウェイ

- 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 の JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールイベントを作成する方法を示します。cron 式 EventBridge 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 を設定します。この例では、Lambda JavaScript ランタイム を使用して Lambda 関数を作成しますAPI。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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

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
- EventBridge
- Lambda
- Amazon SNS

サーバーレスサンプル

Lambda 関数の 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda 関数の Amazon RDS データベースに接続する JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
```

```
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

を使用して Lambda 関数の Amazon RDS データベースに接続する TypeScript。

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
 DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {
```

```
    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
 string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
```

```
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で Kinesis イベントを消費する JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};
```

```
async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

を使用して Lambda で Kinesis イベントを消費する TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};
```

```
async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で DynamoDB イベントを消費する JavaScript。

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

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

を使用して Lambda で DynamoDB イベントを消費する TypeScript。

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

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で Amazon DocumentDB イベントを消費する JavaScript。

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

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
```

```
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

を使用した Lambda での Amazon DocumentDB イベントの消費 TypeScript

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

console.log('Loading function');

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

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

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に記録します。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で Amazon MSKイベントを消費する JavaScript。

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
      })
    }
}
```

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 を呼び出しAPI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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で S3 イベントを消費する JavaScript。

```
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);
    }
};
```

を使用して Lambda で S3 イベントを消費する TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
```

```
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

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信する 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda でSNSイベントを消費する JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
```

```
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

を使用して Lambda でSNSイベントを消費する TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Amazon SQSトリガーから Lambda 関数を呼び出す

次のコード例は、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信する Lambda 関数を実装する方法を示していますSQS。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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda でSQSイベントを消費する JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

を使用して Lambda でSQSイベントを消費する TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";
```

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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;
}
```

を使用して Lambda で Kinesis バッチアイテムの失敗をレポートする TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";
```

```
const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で DynamoDB バッチアイテムの失敗をレポートする JavaScript。

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

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

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

を使用した Lambda での DynamoDB バッチアイテムの失敗のレポート TypeScript。

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

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

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

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

Amazon SQSトリガーを使用した Lambda 関数のバッチアイテム失敗のレポート

次のコード例は、 SQSキュー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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda でSQSバッチアイテムの失敗をレポートする JavaScript。

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
```

```
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

を使用して Lambda でSQSバッチアイテムの失敗をレポートする TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

for JavaScript (v3) を使用した Amazon Lex の例 SDK

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。 Amazon Lex

「シナリオ」は、1 つのサービス内から、または他の AWS 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

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 の JavaScript (v3)

Amazon Lex を使用してウェブアプリケーション内に Chatbot APIを作成し、ウェブサイトの訪問者をエンゲージさせる方法を示します。

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、 AWS SDK for JavaScript デ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

for JavaScript (v3) SDK を使用した Amazon MSKの例

次のコード例は、Amazon 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MSK。

各例には、完全なソースコードへのリンクが含まれています。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

- サーバーレスサンプル

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に記録します。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で Amazon MSKイベントを消費する JavaScript。

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
```

```
      })
    }
}
```

for JavaScript (v3) を使用した Amazon Personalize SDK 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

- アクション

アクション

`CreateBatchInferenceJob`

次のコード例は、`CreateBatchInferenceJob` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchInferenceJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
```

```
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch inference job's parameters.

export const createBatchInferenceJobParam = {
  jobName: 'JOB_NAME',
  jobInput: {         /* required */
    s3DataSource: {   /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional integer*/
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchInferenceJobCommand(createBatchInferenceJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- API 詳細については、「 リファレンスCreateBatchInferenceJob」の「」を参照してください。AWS SDK for JavaScript API

`CreateBatchSegmentJob`

次の例は、CreateBatchSegmentJob を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: 'NAME',
  jobInput: {         /* required */
    s3DataSource: {   /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchSegmentJobCommand(createBatchSegmentJobParam));
    console.log("Success", response);
```

```
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- API 詳細については、「 リファレンスCreateBatchSegmentJob」の「」を参照してください。 AWS SDK for JavaScript API

`CreateCampaign`

次のコード例は、`CreateCampaign` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateCampaign」の「」を参照してください。 AWS SDK for JavaScript API

`CreateDataset`

次の例は、`CreateDataset` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateDataset」の「」を参照してください。 AWS SDK for JavaScript API

`CreateDatasetExportJob`

次のコード例は、CreateDatasetExportJob を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateDatasetExportJob」の「」を参照してください。AWS SDK for JavaScript API

`CreateDatasetGroup`

次の例は、CreateDatasetGroup を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateDatasetGroup」の「」を参照してください。AWS SDK for JavaScript API

`CreateDatasetImportJob`

次の例は、`CreateDatasetImportJob` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateDatasetImportJob」の「」を参照してください。AWS SDK for JavaScript API

`CreateEventTracker`

次のコード例は、`CreateEventTracker` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateEventTracker」の「」を参照してください。 AWS SDK for JavaScript API

`CreateFilter`

次のコード例は、`CreateFilter` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateFilter」の「」を参照してください。 AWS SDK for JavaScript API

`CreateRecommender`

次の例は、`CreateRecommender` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateRecommender」の「」を参照してください。 AWS SDK for JavaScript API

`CreateSchema`

次のコード例は、CreateSchema を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
```

```
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema /* required */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

ドメインを使用して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 'fs';
```

```
let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema, /* required */
  domain: 'DOMAIN'   /* required for a domain dataset group, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createDomainSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- API 詳細については、「 リファレンスCreateSchema」の「」を参照してください。 AWS SDK for JavaScript API

`CreateSolution`

次のコード例は、CreateSolution を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateSolution」の「」を参照してください。 AWS SDK for JavaScript API

`CreateSolutionVersion`

次のコード例は、`CreateSolutionVersion` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスCreateSolutionVersion」の「」を参照してください。AWS SDK for JavaScript API

JavaScript (v3) SDK用の 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

アクション

`PutEvents`

次のコード例は、`PutEvents` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
 Replace it with your sentAt timestamp in UNIX format.
```

```
// Set put events parameters.
var putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- API 詳細については、「 リファレンスPutEvents」の「」を参照してください。 AWS SDK for JavaScript API

PutItems

次のコード例は、PutItems を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Get service clients module and commands using ES6 syntax.
import { PutItemsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put items parameters. For string properties and values, use the \
 character to escape quotes.
var putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /*  required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
 "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- API 詳細については、「 リファレンスPutItems」の「」を参照してください。 AWS SDK for JavaScript API

`PutUsers`

次のコード例は、PutUsers を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Get service clients module and commands using ES6 syntax.
import { PutUsersCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put users parameters. For string properties and values, use the \
 character to escape quotes.
var putUsersParam = {
  datasetArn: "DATASET_ARN",
  users: [
    {
      userId: "USER_ID",
      properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
```

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

- API 詳細については、「 リファレンスPutUsers」の「」を参照してください。 AWS SDK for JavaScript API

SDK for JavaScript (v3) を使用した Amazon Personalize ランタイムの例

次のコード例は、Amazon Personalize Runtime 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスGetPersonalizedRanking」の「」を参照してください。 AWS SDK for JavaScript API

`GetRecommendations`

次のコード例は、GetRecommendations を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// 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 詳細については、「 リファレンスGetRecommendations」の「」を参照してください。AWS SDK for JavaScript API

for JavaScript (v3) を使用した Amazon Pinpoint SDK 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

- アクション

アクション

`SendMessages`

次のコード例は、SendMessages を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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.
var subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
```

```
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
 using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding for the subject line and message body of the email.
var charset = "UTF-8";

const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

const run = async () => {
  try {
    const { MessageResponse } = await pinClient.send(
```

```
      new SendMessagesCommand(params),
    );

    if (!MessageResponse) {
      throw new Error("No message response.");
    }

    if (!MessageResponse.Result) {
      throw new Error("No message result.");
    }

    const recipientResult = MessageResponse.Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    } else {
      console.log(recipientResult.MessageId);
    }
  } catch (err) {
    console.log(err.message);
  }
};

run();
```

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.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderID";

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};
```

```
const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"],
    );
  } catch (err) {
    console.log(err);
  }
};
run();
```

- API 詳細については、「 リファレンスSendMessages」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

E メールメッセージを送信します。

```
"use strict";

const AWS = require("aws-sdk");

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
```

```
// the sandbox, this address also has to be verified.
var toAddress = "recipient@example.com";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https:\/\/aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();
```

```
// Specify the parameters to pass to the API.
var params = {
  ApplicationId: appId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: senderAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

//Try to send the email.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
  } else {
    console.log(
      "Email sent! Message ID: ",
      data["MessageResponse"]["Result"][toAddress]["MessageId"]
    );
  }
});
```

SMS メッセージを送信します。

```
"use strict";

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

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";

// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
var originationNumber = "+12065550199";

// The recipient's phone number.  For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
```

```
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderID";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

//Try to send the message.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
    // Otherwise, show the unique ID for the message.
  } else {
    console.log(
```

```
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  }
});
```

- API 詳細については、「 リファレンスSendMessages」の「」を参照してください。 AWS SDK for JavaScript API

JavaScript (v3) に を使用する Amazon Polly の例 SDK

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。 Amazon Polly

「シナリオ」は、1 つのサービス内から、または他の AWS 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の 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、「」の「 GitHub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

for JavaScript (v3) SDK を使用した Amazon RDSの例

次のコード例は、Amazon 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RDS。

「シナリオ」は、1 つのサービス内から、または他の AWS 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の 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) 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SES。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 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 を使用して、フィルタリングされた作業項目の E メールレポートを送信しますSES。
- 付属の 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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

サーバーレスサンプル

Lambda 関数の 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda 関数の Amazon RDS データベースに接続する JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
```

```
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

を使用して Lambda 関数の Amazon RDS データベースに接続する TypeScript。

```


import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
 DB settings are not null or undefined,
```

```
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
 string }> => {
```

```
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

for JavaScript (v3) を使用した Amazon RDS Data Service SDK 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の 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) 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SES。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 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 を使用して、フィルタリングされた作業項目の E メールレポートを送信しますSES。
- 付属の 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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

for JavaScript (v3) を使用した Amazon Redshift SDK 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

- アクション

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 };
```

クラスターを作成する。

```
// 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 詳細については、「 リファレンスCreateCluster」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteCluster`

次のコード例は、DeleteCluster を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 };
```

クラスターを作成する。

```
// 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 詳細については、「 リファレンスDeleteCluster」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeClusters`

次の例は、`DescribeClusters` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDescribeClusters」の「」を参照してください。 AWS SDK for JavaScript API

`ModifyCluster`

次の例は、`ModifyCluster` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスModifyCluster」の「」を参照してください。 AWS SDK for JavaScript API

for JavaScript (v3) を使用した Amazon Rekognition SDK の例

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。 Amazon Rekognition

「シナリオ」は、1 つのサービス内から、または他の AWS 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の 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セットアップと実行の手順については、「」の詳細な例を参照してください GitHub。

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 ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition

- Amazon S3
- Amazon SNS

イメージPPE内の検出

次のコード例は、Amazon Rekognition を使用して画像内の個人用保護具 (PPE) を検出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 の JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Simple Storage Service (Amazon S3PPE) バケットにあるイメージ内の個人用保護具 () を検出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。Amazon S3 アプリケーションは、結果を Amazon DynamoDB テーブルに保存し、Amazon Simple Email Service (Amazon) 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SES。

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 PPEを使用して のイメージを分析します。 Amazon Rekognition
- Amazon の E メールアドレスを確認しますSES。
- 結果で DynamoDB テーブルを更新します。
- Amazon を使用して E メール通知を送信しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3
- 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 を使用してイメージ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 の 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。アプリは、Amazon Simple Email Service (Amazon) 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SES。

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 の E メールアドレスを確認しますSES。
- Amazon を使用して E メール通知を送信しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

動画内の人物や物体を検出する

次のコード例は、Amazon Rekognition を使用してビデオ内のユーザーとオブジェクトを検出する方法を示しています。

SDK の JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Simple Storage Service (Amazon S3) バケットにあるビデオ内の顔やオブジェクトを検出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。アプリは、Amazon Simple Email Service (Amazon) 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SES。

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 PPEを使用して のイメージを分析します。 Amazon Rekognition
- Amazon の E メールアドレスを確認しますSES。

- Amazon を使用して E メール通知を送信しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript (v3) に を使用する Amazon S3 の例 SDK

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。 Amazon S3

「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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new S3Client({});

export const helloS3 = async () => {
  const command = new ListBucketsCommand({});

  const { Buckets } = await client.send(command);
  console.log("Buckets: ");
  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
  return Buckets;
};
```

- API 詳細については、「 リファレンスListBuckets」の「」を参照してください。 AWS SDK for JavaScript API

トピック

- 基礎
- アクション
- シナリオ
- サーバーレスサンプル

基礎

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 "url";
import { readdirSync, readFileSync, writeFileSync } from "fs";

// Local helper utils.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

import {
  S3Client,
  CreateBucketCommand,
  PutObjectCommand,
  ListObjectsCommand,
  CopyObjectCommand,
  GetObjectCommand,
  DeleteObjectsCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
```

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た GitHub リポジトリにローカルです。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 } from "@inquirer/prompts";

export class Prompter {
  /**
```

```
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  select(options) {
    return select(options);
  }

  /**
   * @param {{ message: string }} options
   */
  input(options) {
    return input(options);
  }

  /**
   * @param {string} prompt
   */
  checkContinue = async (prompt = "") => {
    const prefix = prompt && prompt + " ";
    let ok = await this.confirm({
      message: `${prefix}Continue?`,
    });
    if (!ok) throw new Error("Exiting...");
  };

  /**
   * @param {{ message: string }} options
   */
  confirm(options) {
    return confirm(options);
  }

  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  checkbox(options) {
    return checkbox(options);
  }
}

export const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};
```

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 (let 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 ListObjects に 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 コマンドを使用します。

```
export const copyFileFromBucket = async ({ destinationBucket }) => {
  const proceed = await prompter.confirm({
    message: "Would you like to copy an object from another bucket?",
  });

  if (!proceed) {
    return;
  } else {
    const copy = async () => {
      try {
        const sourceBucket = await prompter.input({
          message: "Enter source bucket name:",
        });
        const sourceKey = await prompter.input({
          message: "Enter source key:",
        });
        const destinationKey = await prompter.input({
          message: "Enter destination key:",
        });

        const command = new CopyObjectCommand({
          Bucket: destinationBucket,
          CopySource: `${sourceBucket}/${sourceKey}`,
          Key: destinationKey,
```

```
        });
        await s3Client.send(command);
        await copyFileFromBucket({ destinationBucket });
      } catch (err) {
        console.error(`Copy error.`);
        console.error(err);
        const retryAnswer = await prompter.confirm({ message: "Try again?" });
        if (retryAnswer) {
          await copy();
        }
      }
    };
    await copy();
  }
};
```

バケットから複数のオブジェクトを取得するSDK方法はありません。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 (let content of Contents) {
    const obj = await s3Client.send(
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
    );
    writeFileSync(
      `${path}/${content.Key}`,
      await obj.Body.transformToByteArray(),
    );
  }
  console.log("Files downloaded successfully.\n");
};
```

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 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

オブジェクトをコピーします。

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

const client = new S3Client({});

export const main = async () => {
  const command = new CopyObjectCommand({
    CopySource: "SOURCE_BUCKET/SOURCE_OBJECT_KEY",
    Bucket: "DESTINATION_BUCKET",
    Key: "NEW_OBJECT_KEY",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- API 詳細については、「 リファレンスCopyObject」の「」を参照してください。 AWS SDK for JavaScript API

`CreateBucket`

次のコード例は、`CreateBucket` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケットを作成します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new CreateBucketCommand({
    // The name of the bucket. Bucket names are unique and have several other
 constraints.
    // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
bucketnamingrules.html
    Bucket: "bucket-name",
  });

  try {
    const { Location } = await client.send(command);
    console.log(`Bucket created with location ${Location}`);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスCreateBucket」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteBucket`

次の例は、`DeleteBucket` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケットを削除します。

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

const client = new S3Client({});

// Delete a bucket.
export const main = async () => {
  const command = new DeleteBucketCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスDeleteBucket」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteBucketPolicy`

次の例は、`DeleteBucketPolicy` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケット ポリシーを削除します。

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

const client = new S3Client({});

// This will remove the policy from the bucket.
export const main = async () => {
  const command = new DeleteBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスDeleteBucketPolicy」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteBucketWebsite`

次のコード例は、`DeleteBucketWebsite` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケットからウェブサイト設定を削除します。

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

const client = new S3Client({});

// Disable static website hosting on the bucket.
export const main = async () => {
  const command = new DeleteBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスDeleteBucketWebsite」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteObject`

次のコード例は、`DeleteObject` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

オブジェクトを削除します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectCommand({
    Bucket: "test-bucket",
    Key: "test-key.txt",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- API 詳細については、「 リファレンスDeleteObject」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteObjects`

次の例は、`DeleteObjects` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

複数のオブジェクトを削除します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectsCommand({
    Bucket: "test-bucket",
    Delete: {
      Objects: [{ Key: "object1.txt" }, { Key: "object2.txt" }],
    },
  });

  try {
    const { Deleted } = await client.send(command);
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted
 objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- API 詳細については、「 リファレンスDeleteObjects」の「」を参照してください。 AWS SDK for JavaScript API

`GetBucketAcl`

次のコード例は、`GetBucketAcl` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

アクセスACL許可を取得します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketAclCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスGetBucketAcl」の「」を参照してください。 AWS SDK for JavaScript API

`GetBucketCors`

次の例は、`GetBucketCors` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケットのCORSポリシーを取得します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketCorsCommand({
    Bucket: "test-bucket",
  });

  try {
    const { CORSRules } = await client.send(command);
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `\n${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders.join(" ")}`,
        `\nAllowedMethods: ${cr.AllowedMethods.join(" ")}`,
        `\nAllowedOrigins: ${cr.AllowedOrigins.join(" ")}`,
        `\nExposeHeaders: ${cr.ExposeHeaders.join(" ")}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスGetBucketCors」の「」を参照してください。 AWS SDK for JavaScript API

GetBucketPolicy

次のコード例は、GetBucketPolicy を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケットポリシーを取得します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const { Policy } = await client.send(command);
    console.log(JSON.parse(Policy));
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスGetBucketPolicy」の「」を参照してください。 AWS SDK for JavaScript API

GetBucketWebsite

次の例は、GetBucketWebsite を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

ウェブサイト設定を取得します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const { ErrorDocument, IndexDocument } = await client.send(command);
    console.log(
      `Your bucket is set up to host a website. It has an error document:`,
      `${ErrorDocument.Key}, and an index document: ${IndexDocument.Suffix}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- API 詳細については、「 リファレンスGetBucketWebsite」の「」を参照してください。 AWS SDK for JavaScript API

`GetObject`

次の例は、`GetObject` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

オブジェクトをダウンロードします。

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

const client = new S3Client({});

export const main = async () => {
  const command = new GetObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
  });

  try {
    const response = await client.send(command);
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスGetObject」の「」を参照してください。 AWS SDK for JavaScript API

`GetObjectLockConfiguration`

次の例は、`GetObjectLockConfiguration` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { fileURLToPath } from "url";
import {
  GetObjectLockConfigurationCommand,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new GetObjectLockConfigurationCommand({
    Bucket: bucketName,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
  });

  try {
    const { ObjectLockConfiguration } = await client.send(command);
    console.log(`Object Lock Configuration: ${ObjectLockConfiguration}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

- API 詳細については、「 リファレンスGetObjectLockConfiguration」の「」を参照してください。 AWS SDK for JavaScript API

GetObjectRetention

次のコード例は、GetObjectRetention を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { fileURLToPath } from "url";
import { GetObjectRetentionCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const { Retention } = await client.send(command);
    console.log(`Object Retention Settings: ${Retention.Status}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- API 詳細については、「 リファレンスGetObjectRetention」の「」を参照してください。 AWS SDK for JavaScript API

`ListBuckets`

次のコード例は、`ListBuckets` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケットを一覧表示します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new ListBucketsCommand({});

  try {
    const { Owner, Buckets } = await client.send(command);
    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`,
    );
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスListBuckets」の「」を参照してください。 AWS SDK for JavaScript API

`ListObjectsV2`

次の例は、`ListObjectsV2` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケット内のすべてのオブジェクトを一覧表示します。複数のオブジェクトがある場合、IsTruncated と NextContinuationToken を使用してリスト全体を反復処理します。

```
import {
  S3Client,
  // This command supersedes the ListObjectsCommand and is the recommended way to
 list objects.
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListObjectsV2Command({
    Bucket: "my-bucket",
    // The default and maximum number of keys returned is 1000. This limits it to
    // one for demonstration purposes.
    MaxKeys: 1,
  });

  try {
    let isTruncated = true;

    console.log("Your bucket contains the following objects:\n");
    let contents = "";

    while (isTruncated) {
```

```
      const { Contents, IsTruncated, NextContinuationToken } =
        await client.send(command);
      const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
      contents += contentsList + "\n";
      isTruncated = IsTruncated;
      command.input.ContinuationToken = NextContinuationToken;
    }
    console.log(contents);
  } catch (err) {
    console.error(err);
  }
};
```

- API 詳細については、「 リファレンス」のListObjectsV2」を参照してください。 AWS SDK for JavaScript API

`PutBucketAcl`

次の例は、PutBucketAcl を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケット を配置しますACL。

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

const client = new S3Client({});

// Most Amazon S3 use cases don't require the use of access control lists (ACLs).
// We recommend that you disable ACLs, except in unusual circumstances where
// you need to control access for each object individually.
// Consider a policy instead. For more information see https://docs.aws.amazon.com/
AmazonS3/latest/userguide/bucket-policies.html.
export const main = async () => {
  // Grant a user READ access to a bucket.
```

```
  const command = new PutBucketAclCommand({
    Bucket: "test-bucket",
    AccessControlPolicy: {
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
 AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
            ID: "canonical-id-1",
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "FULL_CONTROL",
        },
      ],
      Owner: {
        ID: "canonical-id-2",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスPutBucketAcl」の「」を参照してください。 AWS SDK for JavaScript API

PutBucketCors

次のコード例は、PutBucketCors を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

CORS ルールを追加します。

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

const client = new S3Client({});

// By default, Amazon S3 doesn't allow cross-origin requests. Use this command
// to explicitly allow cross-origin requests.
export const main = async () => {
  const command = new PutBucketCorsCommand({
    Bucket: "test-bucket",
    CORSConfiguration: {
      CORSRules: [
        {
          // Allow all headers to be sent to this bucket.
          AllowedHeaders: ["*"],
          // Allow only GET and PUT methods to be sent to this bucket.
          AllowedMethods: ["GET", "PUT"],
          // Allow only requests from the specified origin.
          AllowedOrigins: ["https://www.example.com"],
          // Allow the entity tag (ETag) header to be returned in the response. The
 ETag header
          // The entity tag represents a specific version of the object. The ETag
 reflects
          // changes only to the contents of an object, not its metadata.
          ExposeHeaders: ["ETag"],
          // How long the requesting browser should cache the preflight response.
 After
          // this time, the preflight request will have to be made again.
          MaxAgeSeconds: 3600,
        },
      ],
    },
  });
```

```
  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスPutBucketCors」の「」を参照してください。 AWS SDK for JavaScript API

PutBucketPolicy

次の例は、PutBucketPolicy を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

ポリシーを追加します。

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

const client = new S3Client({});

export const main = async () => {
  const command = new PutBucketPolicyCommand({
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "AllowGetObject",
          // Allow this particular user to call GetObject on any object in this
 bucket.
          Effect: "Allow",
```

```
          Principal: {
            AWS: "arn:aws:iam::ACCOUNT-ID:user/USERNAME",
          },
          Action: "s3:GetObject",
          Resource: "arn:aws:s3:::BUCKET-NAME/*",
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: "BUCKET-NAME",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスPutBucketPolicy」の「」を参照してください。 AWS SDK for JavaScript API

`PutBucketWebsite`

次のコード例は、`PutBucketWebsite` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

ウェブサイト設定を設定します。

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

```
const client = new S3Client({});

// Set up a bucket as a static website.
// The bucket needs to be publicly accessible.
export const main = async () => {
  const command = new PutBucketWebsiteCommand({
    Bucket: "test-bucket",
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
        // A suffix that is appended to a request that is for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスPutBucketWebsite」の「」を参照してください。 AWS SDK for JavaScript API

PutObject

次のコード例は、PutObject を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

オブジェクトをアップロードします。

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

const client = new S3Client({});

export const main = async () => {
  const command = new PutObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
    Body: "Hello S3!",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 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、「 リファレンスPutObject」の「」を参照してください。 AWS SDK for JavaScript API

`PutObjectLegalHold`

次の例は、`PutObjectLegalHold` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { fileURLToPath } from "url";
import { PutObjectLegalHoldCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    LegalHold: {
      // Set the status to 'ON' to place a legal hold on the object.
      // Set the status to 'OFF' to remove the legal hold.
      Status: "ON",
    },
    // Optionally, you can provide additional parameters
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object legal hold status: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

```
// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- API 詳細については、「 リファレンスPutObjectLegalHold」の「」を参照してください。 AWS SDK for JavaScript API

`PutObjectLockConfiguration`

次のコード例は、PutObjectLockConfiguration を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

バケットのオブジェクトロック設定を指定します。

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
    // Optionally, you can provide additional parameters
```

```
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Lock Configuration updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

バケットのデフォルトの保存期間を設定します。

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

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
      Rule: {
        DefaultRetention: {
          Mode: "GOVERNANCE",
          Years: 3,
```

```
        },
      },
    },
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Default Object Lock Configuration updated: ${response.
$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```

- API 詳細については、「 リファレンスPutObjectLockConfiguration」の「」を参照してください。 AWS SDK for JavaScript API

`PutObjectRetention`

次のコード例は、`PutObjectRetention` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { fileURLToPath } from "url";
import { PutObjectRetentionCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    BypassGovernanceRetention: false,
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    Retention: {
      Mode: "GOVERNANCE", // or "COMPLIANCE"
      RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Retention settings updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- API 詳細については、「 リファレンスPutObjectRetention」の「」を参照してください。 AWS SDK for JavaScript API

シナリオ

署名付き を作成する URL

次のコード例は、Amazon S3 URLの署名付き 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。 Amazon S3

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 "https";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
  return formatUrl(signedUrlObject);
};
```

```
const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

function put(url, data) {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          resolve(responseBody);
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
}

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.txt";

  // There are two ways to generate a presigned URL.
  // 1. Use createPresignedUrl without the S3 client.
  // 2. Use getSignedUrl in conjunction with the S3 client and GetObjectCommand.
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });
```

```
    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    // After you get the presigned URL, you can provide your own file
    // data. Refer to put() above.
    console.log("Calling PUT using presigned URL without client");
    await put(noClientUrl, "Hello World");

    console.log("Calling PUT using presigned URL with client");
    await put(clientUrl, "Hello World");

    console.log("\nDone. Check your S3 console.");
  } catch (err) {
    console.error(err);
  }
};
```

署名付き URLを作成して、バケットからオブジェクトを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 });
};

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.jpg";

  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    console.log("Presigned URL without client");
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (err) {
    console.error(err);
  }
};
```

- 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 の 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セットアップと実行の手順については、「」の詳細な例を参照してください GitHub。

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 ゲートウェイ
- 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

次のコードは、 を呼び出す関連する React コンポーネントです AWS SDK。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、前の GitHub リンクにあります。

```
import { useEffect, useState } from "react";
```

```
import {
  ListObjectsCommand,
  ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";

function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
 role to the pool,
      // and grant the role access to the 's3:GetObject' action.
      //
      // You'll also need to configure the CORS settings on the bucket to allow
 traffic from
      // this example site. Here's an example configuration that allows all origins.
 Don't
      // do this in production.
      //[
      //  {
      //    "AllowedHeaders": ["*"],
      //    "AllowedMethods": ["GET"],
      //    "AllowedOrigins": ["*"],
      //    "ExposeHeaders": [],
      //  },
      //]
      //
      credentials: fromCognitoIdentityPool({
        clientConfig: { region: "us-east-1" },
        identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
      }),
    });
    const command = new ListObjectsCommand({ Bucket: "bucket-name" });
    client.send(command).then(({ Contents }) => setObjects(Contents || []));
  }, []);
```

```
  return (
    <div className="App">
      {objects.map((o) => (
        <div key={o.ETag}>{o.Key}</div>
      ))}
    </div>
  );
}

export default App;
```

- API 詳細については、「 リファレンスListObjects」の「」を参照してください。 AWS SDK for JavaScript API

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 の 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) をストレージとして使用し、Amazon Simple Notification Service (Amazon SQS) トピックにサブスクライブされている Amazon Simple Queue Service (Amazon SNS) キューをポーリングする通知に使用します。

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、「」の詳細な例を参照してくださいGitHub。

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

イメージPPE内の検出

次のコード例は、Amazon Rekognition を使用してイメージ内の個人用保護具 (PPE) を検出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 の JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Simple Storage Service (Amazon S3PPE) バケットにあるイメージ内の個人用保護具 () を検出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。Amazon S3 アプリケーションは、結果を Amazon DynamoDB テーブルに保存し、Amazon Simple Email Service (Amazon) 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SES。

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 PPEを使用して のイメージを分析します。 Amazon Rekognition
- Amazon の E メールアドレスを確認しますSES。
- 結果で DynamoDB テーブルを更新します。
- Amazon を使用して E メール通知を送信しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3
- 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 を使用してイメージ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 の 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) を使用して、結果が記載された E メール通知を管理者に送信しますSES。

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 の E メールアドレスを確認しますSES。
- Amazon を使用して E メール通知を送信しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

オブジェクトのリーガルホールド設定を取得する

次のコード例は、S3 バケットのリーガルホールド設定を取得する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { fileURLToPath } from "url";
import { GetObjectLegalHoldCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
```

```
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const response = await client.send(command);
    console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "DOC-EXAMPLE-BUCKET", "OBJECT_KEY");
}
```

- API 詳細については、「 リファレンスGetObjectLegalHold」の「」を参照してください。 AWS SDK for JavaScript API

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

index.js - ワークフローのエントリポイント。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 シナリオ、、および の実装の詳細を確認するには ScenarioInput、ScenarioOutput「」を参照してください ScenarioAction。

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "./welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  confirmSetLegalHoldFileEnabled,
  confirmSetLegalHoldFileRetention,
  confirmSetRetentionPeriodFileEnabled,
  confirmSetRetentionPeriodFileRetention,
  confirmUpdateLockPolicy,
  confirmUpdateRetention,
  createBuckets,
  createBucketsAction,
  populateBuckets,
  populateBucketsAction,
  setLegalHoldFileEnabledAction,
  setLegalHoldFileRetentionAction,
  setRetentionPeriodFileEnabledAction,
  setRetentionPeriodFileRetentionAction,
  updateLockPolicy,
  updateLockPolicyAction,
  updateRetention,
  updateRetentionAction,
} from "./setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
```

```
  deploy: new scenarios.Scenario(
    "S3 Object Locking - Deploy",
    [
      welcome(scenarios),
      welcomeContinue(scenarios),
      exitOnFalse(scenarios, "welcomeContinue"),
      createBuckets(scenarios),
      confirmCreateBuckets(scenarios),
      exitOnFalse(scenarios, "confirmCreateBuckets"),
      createBucketsAction(scenarios, client),
      updateRetention(scenarios),
      confirmUpdateRetention(scenarios),
      exitOnFalse(scenarios, "confirmUpdateRetention"),
      updateRetentionAction(scenarios, client),
      populateBuckets(scenarios),
      confirmPopulateBuckets(scenarios),
      exitOnFalse(scenarios, "confirmPopulateBuckets"),
      populateBucketsAction(scenarios, client),
      updateLockPolicy(scenarios),
      confirmUpdateLockPolicy(scenarios),
      exitOnFalse(scenarios, "confirmUpdateLockPolicy"),
      updateLockPolicyAction(scenarios, client),
      confirmSetLegalHoldFileEnabled(scenarios),
      setLegalHoldFileEnabledAction(scenarios, client),
      confirmSetRetentionPeriodFileEnabled(scenarios),
      setRetentionPeriodFileEnabledAction(scenarios, client),
      confirmSetLegalHoldFileRetention(scenarios),
      setLegalHoldFileRetentionAction(scenarios, client),
      confirmSetRetentionPeriodFileRetention(scenarios),
      setRetentionPeriodFileRetentionAction(scenarios, client),
      saveState,
    ],
    initialState,
  ),
  demo: new scenarios.Scenario(
    "S3 Object Locking - Demo",
    [loadState, replAction(scenarios, client)],
    initialState,
  ),
  clean: new scenarios.Scenario(
    "S3 Object Locking - Destroy",
    [
      loadState,
      confirmCleanup(scenarios),
```

```
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
      ],
      initialState,
    ),
  };
};

// Call function if run directly
import { fileURLToPath } from "url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios);
}
```

welcome.steps.js - ウェルカムメッセージをコンソールに出力します。

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    `Welcome to the Amazon Simple Storage Service (S3) Object Locking Workflow
 Scenario. For this workflow, we will use the AWS SDK for JavaScript to create
 several S3 buckets and files to demonstrate working with S3 locking features.`,
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
```

```
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };
```

setup.steps.js - バケット、オブジェクト、ファイル設定をデプロイします。

```
import {
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const bucketPrefix = "js-object-locking";

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    `The following buckets will be created:
         ${bucketPrefix}-no-lock with object lock False.
```

```
          ${bucketPrefix}-lock-enabled with object lock True.
          ${bucketPrefix}-retention-after-creation with object lock False.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const noLockBucketName = `${bucketPrefix}-no-lock`;
    const lockEnabledBucketName = `${bucketPrefix}-lock-enabled`;
    const retentionBucketName = `${bucketPrefix}-retention-after-creation`;

    await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
    await client.send(
      new CreateBucketCommand({
        Bucket: lockEnabledBucketName,
        ObjectLockEnabledForBucket: true,
      }),
    );
    await client.send(new CreateBucketCommand({ Bucket: retentionBucketName }));

    state.noLockBucketName = noLockBucketName;
    state.lockEnabledBucketName = lockEnabledBucketName;
    state.retentionBucketName = retentionBucketName;
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    `The following test files will be created:
```

```
          file0.txt in ${bucketPrefix}-no-lock.
          file1.txt in ${bucketPrefix}-no-lock.
          file0.txt in ${bucketPrefix}-lock-enabled.
          file1.txt in ${bucketPrefix}-lock-enabled.
          file0.txt in ${bucketPrefix}-retention-after-creation.
          file1.txt in ${bucketPrefix}-retention-after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
```

```
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateRetention = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateRetention",
    `A bucket can be configured to use object locking with a default retention
 period.
   A default retention period will be configured for ${bucketPrefix}-retention-
after-creation.`,
    { preformatted: true },
  );
```

```
/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateRetention",
    "Configure default retention period?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateRetentionAction", async (state) => {
    await client.send(
      new PutBucketVersioningCommand({
        Bucket: state.retentionBucketName,
        VersioningConfiguration: {
          MFADelete: MFADeleteStatus.Disabled,
          Status: BucketVersioningStatus.Enabled,
        },
      }),
    );

    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.retentionBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
          Rule: {
            DefaultRetention: {
              Mode: "GOVERNANCE",
              Years: 1,
            },
          },
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
```

```
 */
const updateLockPolicy = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateLockPolicy",
    `Object lock policies can also be added to existing buckets.
   An object lock policy will be added to ${bucketPrefix}-lock-enabled.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateLockPolicy = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateLockPolicy",
    "Add object lock policy?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateLockPolicyAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateLockPolicyAction", async (state) => {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.lockEnabledBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileEnabled",
    (state) =>
```

```
      `Would you like to add a legal hold to file0.txt in
 ${state.lockEnabledBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileEnabledAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.lockEnabledBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetRetentionPeriodFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileEnabled",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.lockEnabledBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
```

```
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileEnabledAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
        }),
      );
      console.log(
        `Set retention for file1.txt in ${state.lockEnabledBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.retentionBucketName}?`,
    {
      type: "confirm",
```

```
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileRetentionAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.retentionBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.retentionBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileRetention },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmSetRetentionPeriodFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileRetention",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.retentionBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
```

```
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileRetentionAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.retentionBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
          BypassGovernanceRetention: true,
        }),
      );
      console.log(
        `Set retention for file1.txt in ${state.retentionBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileRetention },
  );

export {
  createBuckets,
  confirmCreateBuckets,
  createBucketsAction,
  populateBuckets,
  confirmPopulateBuckets,
  populateBucketsAction,
  updateRetention,
  confirmUpdateRetention,
  updateRetentionAction,
  updateLockPolicy,
  confirmUpdateLockPolicy,
  updateLockPolicyAction,
  confirmSetLegalHoldFileEnabled,
  setLegalHoldFileEnabledAction,
  confirmSetRetentionPeriodFileEnabled,
```

```
  setRetentionPeriodFileEnabledAction,
  confirmSetLegalHoldFileRetention,
  setLegalHoldFileRetentionAction,
  confirmSetRetentionPeriodFileRetention,
  setRetentionPeriodFileRetentionAction,
};
```

repl.steps.js - バケット内のファイルを表示および削除します。

```
import {
  ChecksumAlgorithm,
  DeleteObjectCommand,
  GetObjectLegalHoldCommand,
  GetObjectLockConfigurationCommand,
  GetObjectRetentionCommand,
  ListObjectVersionsCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  DELETE_FILE: 2,
  DELETE_FILE_WITH_RETENTION: 3,
  OVERWRITE_FILE: 4,
  VIEW_RETENTION_SETTINGS: 5,
  VIEW_LEGAL_HOLD_SETTINGS: 6,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new scenarios.ScenarioInput(
```

```
    "replChoice",
    `Explore the S3 locking features by selecting one of the following choices`,
    {
      type: "select",
      choices: [
        { name: "List all files in buckets", value: choices.LIST_ALL_FILES },
        { name: "Attempt to delete a file.", value: choices.DELETE_FILE },
        {
          name: "Attempt to delete a file with retention period bypass.",
          value: choices.DELETE_FILE_WITH_RETENTION,
        },
        { name: "Attempt to overwrite a file.", value: choices.OVERWRITE_FILE },
        {
          name: "View the object and bucket retention settings for a file.",
          value: choices.VIEW_RETENTION_SETTINGS,
        },
        {
          name: "View the legal hold settings for a file.",
          value: choices.VIEW_LEGAL_HOLD_SETTINGS,
        },
        { name: "Finish the workflow.", value: choices.EXIT },
      ],
    },
  );

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;
      files.push({ bucket, key: Key, version: VersionId });
    }
  }

  return files;
};
```

```
/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const replAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.noLockBucketName,
        state.lockEnabledBucketName,
        state.retentionBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file:",
        {
          type: "select",
          choices: files.map((file, index) => ({
            name: `${index + 1}: ${file.bucket}: ${file.key} (version: ${
              file.version
            })`,
            value: index,
          })),
        },
      );

      const { replChoice } = state;

      switch (replChoice) {
        case choices.LIST_ALL_FILES: {
          const files = await getAllFiles(client, [
            state.noLockBucketName,
            state.lockEnabledBucketName,
            state.retentionBucketName,
          ]);
          state.replOutput = files
            .map(
              (file) =>
                `${file.bucket}: ${file.key} (version: ${file.version})`,
            )
            .join("\n");
```

```
          break;
        }
        case choices.DELETE_FILE: {
          /** @type {number} */
          const fileToDelete = await fileInput.handle(state);
          const selectedFile = files[fileToDelete];
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.DELETE_FILE_WITH_RETENTION: {
          /** @type {number} */
          const fileToDelete = await fileInput.handle(state);
          const selectedFile = files[fileToDelete];
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
                BypassGovernanceRetention: true,
              }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.OVERWRITE_FILE: {
```

```
          /** @type {number} */
          const fileToOverwrite = await fileInput.handle(state);
          const selectedFile = files[fileToOverwrite];
          try {
            await client.send(
              new PutObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                Body: "New content",
                ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
              }),
            );
            state.replOutput = `Overwrote ${selectedFile.key} in
 ${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to overwrite object ${selectedFile.key} in
 bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.VIEW_RETENTION_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
            const retention = await client.send(
              new GetObjectRetentionCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            const bucketConfig = await client.send(
              new GetObjectLockConfigurationCommand({
                Bucket: selectedFile.bucket,
              }),
            );
            state.replOutput = `Object retention for ${selectedFile.key}
 in ${selectedFile.bucket}: ${retention.Retention?.Mode} until
 ${retention.Retention?.RetainUntilDate?.toISOString()}.
Bucket object lock config for ${selectedFile.bucket} in ${selectedFile.bucket}:
Enabled: ${bucketConfig.ObjectLockConfiguration?.ObjectLockEnabled}
Rule:
 ${JSON.stringify(bucketConfig.ObjectLockConfiguration?.Rule?.DefaultRetention)}`;
```

```
          } catch (err) {
            state.replOutput = `Unable to fetch object lock retention:
 '${err.message}'`;
          }
          break;
        }
        case choices.VIEW_LEGAL_HOLD_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
            const legalHold = await client.send(
              new GetObjectLegalHoldCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Object legal hold for ${selectedFile.key} in
 ${selectedFile.bucket}: Status: ${legalHold.LegalHold?.Status}`;
          } catch (err) {
            state.replOutput = `Unable to fetch legal hold: '${err.message}'`;
          }
          break;
        }
        default:
          throw new Error(`Invalid replChoice: ${replChoice}`);
      }
    },
    {
      whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new scenarios.ScenarioOutput(
          "REPL output",
          (state) => state.replOutput,
          { preformatted: true },
        ),
      },
    },
  );

export { replInput, replAction, choices };
```

clean.steps.js - 作成されたすべてのリソースを破棄します。

```
import {
  DeleteObjectCommand,
  DeleteBucketCommand,
  ListObjectVersionsCommand,
  GetObjectLegalHoldCommand,
  GetObjectRetentionCommand,
  PutObjectLegalHoldCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { noLockBucketName, lockEnabledBucketName, retentionBucketName } =
      state;

    const buckets = [
      noLockBucketName,
      lockEnabledBucketName,
      retentionBucketName,
    ];
```

```
  for (const bucket of buckets) {
    /** @type {import("@aws-sdk/client-s3").ListObjectVersionsCommandOutput} */
    let objectsResponse;

    try {
      objectsResponse = await client.send(
        new ListObjectVersionsCommand({
          Bucket: bucket,
        }),
      );
    } catch (e) {
      if (e instanceof Error && e.name === "NoSuchBucket") {
        console.log("Object's bucket has already been deleted.");
        continue;
      } else {
        throw e;
      }
    }

    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;

      try {
        const legalHold = await client.send(
          new GetObjectLegalHoldCommand({
            Bucket: bucket,
            Key,
            VersionId,
          }),
        );

        if (legalHold.LegalHold?.Status === "ON") {
          await client.send(
            new PutObjectLegalHoldCommand({
              Bucket: bucket,
              Key,
              VersionId,
              LegalHold: {
                Status: "OFF",
              },
            }),
          );
        }
```

```
        } catch (err) {
          console.log(
            `Unable to fetch legal hold for ${Key} in ${bucket}: '${err.message}'`,
          );
        }

        try {
          const retention = await client.send(
            new GetObjectRetentionCommand({
              Bucket: bucket,
              Key,
              VersionId,
            }),
          );

          if (retention.Retention?.Mode === "GOVERNANCE") {
            await client.send(
              new DeleteObjectCommand({
                Bucket: bucket,
                Key,
                VersionId,
                BypassGovernanceRetention: true,
              }),
            );
          }
        } catch (err) {
          console.log(
            `Unable to fetch object lock retention for ${Key} in ${bucket}:
'${err.message}'`,
          );
        }

        await client.send(
          new DeleteObjectCommand({
            Bucket: bucket,
            Key,
            VersionId,
          }),
        );
      }

      await client.send(new DeleteBucketCommand({ Bucket: bucket }));
      console.log(`Delete for ${bucket} complete.`);
    }
```

```
  });

export { confirmCleanup, cleanupAction };
```

- API 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

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

サイズの大きいファイルをアップロードします。

```
import {
  CreateMultipartUploadCommand,
  UploadPartCommand,
  CompleteMultipartUploadCommand,
  AbortMultipartUploadCommand,
  S3Client,
} from "@aws-sdk/client-s3";
```

```
const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

export const main = async () => {
  const s3Client = new S3Client({});
  const bucketName = "test-bucket";
  const key = "multipart.txt";
  const str = createString();
  const buffer = Buffer.from(str, "utf8");

  let uploadId;

  try {
    const multipartUpload = await s3Client.send(
      new CreateMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );

    uploadId = multipartUpload.UploadId;

    const uploadPromises = [];
    // Multipart uploads require a minimum size of 5 MB per part.
    const partSize = Math.ceil(buffer.length / 5);

    // Upload each part.
    for (let i = 0; i < 5; i++) {
      const start = i * partSize;
      const end = start + partSize;
      uploadPromises.push(
        s3Client
          .send(
            new UploadPartCommand({
              Bucket: bucketName,
              Key: key,
              UploadId: uploadId,
              Body: buffer.subarray(start, end),
              PartNumber: i + 1,
            }),
```

```
          )
          .then((d) => {
            console.log("Part", i + 1, "uploaded");
            return d;
          }),
      );
    }

    const uploadResults = await Promise.all(uploadPromises);

    return await s3Client.send(
      new CompleteMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
        UploadId: uploadId,
        MultipartUpload: {
          Parts: uploadResults.map(({ ETag }, i) => ({
            ETag,
            PartNumber: i + 1,
          })),
        },
      }),
    );

    // Verify the output by downloading the file from the Amazon Simple Storage
 Service (Amazon S3) console.
    // Because the output is a 25 MB string, text editors might struggle to open the
 file.
  } catch (err) {
    console.error(err);

    if (uploadId) {
      const abortCommand = new AbortMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
        UploadId: uploadId,
      });

      await s3Client.send(abortCommand);
    }
  }
};
```

サイズの大きいファイルをダウンロードします。

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

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;

export const getObjectRange = ({ bucket, key, start, end }) => {
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

/**
 * @param {string | undefined} contentRange
 */
export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: parseInt(start),
    end: parseInt(end),
    length: parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

// When downloading a large file, you might want to break it down into
// smaller pieces. Amazon S3 accepts a Range header to specify the start
// and end of the byte range to be downloaded.
const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url)),
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
```

```
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    console.log(`Downloading bytes ${nextRange.start} to ${nextRange.end}`);

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
    });

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

export const main = async () => {
  await downloadInChunks({
    bucket: "my-cool-bucket",
    key: "my-cool-object.txt",
  });
};
```

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 を呼び出しAPI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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda で S3 イベントを消費する JavaScript。

```
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);
    }
};
```

を使用して Lambda で S3 イベントを消費する TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
```

```
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

SDK を使用した S3 Glacier の例 JavaScript (v3)

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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

- アクション

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 };
```

ボールトを作成する

```
// 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 詳細については、「 リファレンスCreateVault」の「」を参照してください。 AWS SDK for JavaScript API

SDK JavaScript (v2) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
// Call Glacier to create the vault
glacier.createVault({ vaultName: "YOUR_VAULT_NAME" }, function (err) {
  if (!err) {
    console.log("Created vault!");
  }
});
```

- 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、「 リファレンスCreateVault」の「」を参照してください。 AWS SDK for JavaScript API

`UploadArchive`

次の例は、`UploadArchive` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスUploadArchive」の「」を参照してください。 AWS SDK for JavaScript API

SDK JavaScript (v2) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object and buffer
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
buffer = Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer

var params = { vaultName: "YOUR_VAULT_NAME", body: buffer };
// Call Glacier to upload the archive.
glacier.uploadArchive(params, function (err, data) {
  if (err) {
    console.log("Error uploading archive!", err);
  } else {
    console.log("Archive ID", data.archiveId);
  }
});
```

- 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、「 リファレンスUploadArchive」の「」を参照してください。 AWS SDK for JavaScript API

SageMaker JavaScript (v3) SDK に を使用する 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 SageMaker。

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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

開始方法

こんにち SageMakerは

次のコード例は、 の使用を開始する方法を示しています SageMaker。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  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 詳細については、「 リファレンスListNotebookInstances」の「」を参照してください。AWS SDK for JavaScript API

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

ローカルで指定されたJSON定義を使用して SageMaker 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,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
```

```
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}
```

- API 詳細については、「 リファレンスCreatePipeline」の「」を参照してください。 AWS SDK for JavaScript API

`DeletePipeline`

次のコード例は、`DeletePipeline` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

SageMaker 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 詳細については、「 リファレンスDeletePipeline」の「」を参照してください。 AWS SDK for JavaScript API

`DescribePipelineExecution`

次の例は、`DescribePipelineExecution` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

SageMaker 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;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
```

```
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- API 詳細については、「 リファレンスDescribePipelineExecution」の「」を参照してください。 AWS SDK for JavaScript API

`StartPipelineExecution`

次のコード例は、`StartPipelineExecution` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

SageMaker 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 詳細については、「 リファレンスStartPipelineExecution」の「」を参照してください。 AWS SDK for JavaScript API

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 の「 を使用して SageMaker AWS SDKsパイプラインを作成および実行する」を参照してください。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

次のファイルの抜粋には、 SageMaker 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 "fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
```

```
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
  GetRoleCommand,
  ListPoliciesCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
  GetFunctionCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-
geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
```

```
  GetQueueUrlCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 props
 */
export async function createLambdaExecutionRole({ name, iamClient }) {
  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: { Service: ["lambda.amazonaws.com"] },
            },
          ],
        }),
      }),
    );

  let role = null;

  try {
    const { Role } = await createRole();
    role = Role;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
```

```
      throw caught;
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 Amazon SageMaker.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: [
          "sqs:ReceiveMessage",
          "sqs:DeleteMessage",
          "sqs:GetQueueAttributes",
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents",
          "sagemaker-geospatial:StartVectorEnrichmentJob",
          "sagemaker-geospatial:GetVectorEnrichmentJob",
          "sagemaker:SendPipelineExecutionStepFailure",
          "sagemaker:SendPipelineExecutionStepSuccess",
          "sagemaker-geospatial:ExportVectorEnrichmentJob",
        ],
        Resource: "*",
```

```
      },
      {
        Effect: "Allow",
        // The AWS Lambda function needs permission to pass the pipeline execution
 role to
        // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
 function
        // from elevating privileges. For more information, see:
        // https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_use_passrole.html
        Action: ["iam:PassRole"],
        Resource: `${pipelineExecutionRoleArn}`,
        Condition: {
          StringEquals: {
            "iam:PassedToService": [
              "sagemaker.amazonaws.com",
              "sagemaker-geospatial.amazonaws.com",
            ],
          },
        },
      },
    ],
  };

  const createPolicy = () =>
    iamClient.send(
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
        PolicyName: name,
      }),
    );

  let policy = null;

  try {
    const { Policy } = await createPolicy();
    policy = Policy;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
```

```
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}

/**
 * Attach an AWS IAM policy to an AWS IAM role.
 * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-
iam').IAMClient}} props
 */
export async function attachPolicy({ roleName, policyArn, iamClient }) {
  const attachPolicyCommand = new AttachRolePolicyCommand({
    RoleName: roleName,
    PolicyArn: policyArn,
  });

  await iamClient.send(attachPolicyCommand);
  return {
    cleanUp: async () => {
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: roleName,
          PolicyArn: policyArn,
        }),
      );
    },
  };
}

/**
```

```
 * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon
 SageMaker Geospatial clients
 * in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
 Amazon SageMaker
 * Geospatial client wasn't introduced until v3.221.0.
 * @param {{ name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient }} props
 */
export async function createLambdaLayer({ name, lambdaClient }) {
  const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;
  const { LayerVersionArn, Version } = await lambdaClient.send(
    new PublishLayerVersionCommand({
      LayerName: name,
      Content: {
        ZipFile: Uint8Array.from(readFileSync(layerPath)),
      },
    }),
  );

  return {
    versionArn: LayerVersionArn,
    version: Version,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteLayerVersionCommand({
          LayerName: name,
          VersionNumber: Version,
        }),
      );
    },
  };
}

/**
 * Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
 pipeline
 * execution steps.
 * @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient, layerVersionArn: string}} props
 */
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
```

```
  layerVersionArn,
}) {
  const lambdaPath = `${dirnameFromMetaUrl(
    import.meta.url,
  )}lambda/dist/index.mjs.zip`;

  // If a function of the same name already exists, return that
  // function's ARN instead. By default this is
  // "sagemaker-wkflw-lambda-function", so collisions are
  // unlikely.
  const createFunction = async () => {
    try {
      return await lambdaClient.send(
        new CreateFunctionCommand({
          Code: {
            ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
          },
          Runtime: Runtime.nodejs18x,
          Handler: "index.handler",
          Layers: [layerVersionArn],
          FunctionName: name,
          Role: roleArn,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "ResourceConflictException"
      ) {
        const { Configuration } = await lambdaClient.send(
          new GetFunctionCommand({ FunctionName: name }),
        );
        return Configuration;
      } else {
        throw caught;
      }
    }
  };

  // Function creation fails if the Role is not ready. This retries
  // function creation until it succeeds or it times out.
  const { FunctionArn } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    createFunction,
```

```
  );

  return {
    arn: FunctionArn,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteFunctionCommand({ FunctionName: name }),
      );
    },
  };
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
 props
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
  const s3Path = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../../workflows/sagemaker_pipelines/resources/latlongtest.csv`;

  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "input/sample_data.csv",
      Body: readFileSync(s3Path),
    }),
  );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient, wait:
 (ms: number) => Promise<void>}} props
 */
export async function createSagemakerRole({ name, iamClient, wait }) {
  let role = null;

  const createRole = () =>
    iamClient.send(
```

```
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: {
                Service: [
                  "sagemaker.amazonaws.com",
                  "sagemaker-geospatial.amazonaws.com",
                ],
              },
            },
          ],
        }),
      }),
    );

  try {
    const { Role } = await createRole();
    role = Role;
    // Wait for the role to be ready.
    await wait(10);
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
```

```
  };
}

/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 messages to
 * the Amazon SQS queue.
 * @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
 import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
 */
export async function createSagemakerExecutionPolicy({
  sqsQueueArn,
  lambdaArn,
  iamClient,
  name,
  s3BucketName,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: ["lambda:InvokeFunction"],
        Resource: lambdaArn,
      },
      {
        Effect: "Allow",
        Action: ["s3:*"],
        Resource: [
          `arn:aws:s3:::${s3BucketName}`,
          `arn:aws:s3:::${s3BucketName}/*`,
        ],
      },
      {
        Effect: "Allow",
        Action: ["sqs:SendMessage"],
        Resource: sqsQueueArn,
      },
    ],
  };

  const createPolicy = () =>
    iamClient.send(
```

```
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
        PolicyName: name,
      }),
    );

  let policy = null;

  try {
    const { Policy } = await createPolicy();
    policy = Policy;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}

/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
```

```
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
```

```
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
  const createSqsQueue = () =>
    sqsClient.send(
      new CreateQueueCommand({
        QueueName: name,
        Attributes: {
          DelaySeconds: "5",
          ReceiveMessageWaitTimeSeconds: "5",
          VisibilityTimeout: "300",
        },
      }),
    );

  let queueUrl = null;
  try {
    const { QueueUrl } = await createSqsQueue();
    queueUrl = QueueUrl;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "QueueNameExists") {
      const { QueueUrl } = await sqsClient.send(
        new GetQueueUrlCommand({ QueueName: name }),
      );
      queueUrl = QueueUrl;
```

```
    } else {
      throw caught;
    }
  }

  const { Attributes } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    () =>
      sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: queueUrl,
          AttributeNames: ["QueueArn"],
        }),
      ),
  );

  return {
    queueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl: queueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{
 *   paginateListEventSourceMappings: () => Generator<import('@aws-sdk/client-
lambda').ListEventSourceMappingsCommandOutput>,
 *   lambdaName: string,
 *   queueArn: string,
 *   lambdaClient: import('@aws-sdk/client-lambda').LambdaClient}} props
 */
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
  paginateListEventSourceMappings,
}) {
  let uuid = null;
  const createEvenSourceMapping = () =>
    lambdaClient.send(
```

```
      new CreateEventSourceMappingCommand({
        EventSourceArn: queueArn,
        FunctionName: lambdaName,
      }),
    );

  try {
    const { UUID } = await createEvenSourceMapping();
    uuid = UUID;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceConflictException"
    ) {
      const paginator = paginateListEventSourceMappings(
        { client: lambdaClient },
        {},
      );
      /**
       * @type {import('@aws-sdk/client-lambda').EventSourceMappingConfiguration[]}
       */
      const eventSourceMappings = [];
      for await (const page of paginator) {
        eventSourceMappings.concat(page.EventSourceMappings || []);
      }

      const { Configuration } = await lambdaClient.send(
        new GetFunctionCommand({ FunctionName: lambdaName }),
      );

      uuid = eventSourceMappings.find(
        (mapping) =>
          mapping.EventSourceArn === queueArn &&
          mapping.FunctionArn === Configuration.FunctionArn,
      ).UUID;
    } else {
      throw caught;
    }
  }

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
```

```
          UUID: uuid,
        }),
      );
    },
  };
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{
 *   s3Client: import('@aws-sdk/client-s3').S3Client,
 *   name: string,
 *   paginateListObjectsV2: () => Generator<import('@aws-sdk/client-
s3').ListObjectsCommandOutput>
 * }} props
 */
export async function createS3Bucket({
  name,
  s3Client,
  paginateListObjectsV2,
}) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(
        { client: s3Client },
        { Bucket: name },
      );
      for await (const page of paginator) {
        const objects = page.Contents;
        if (objects) {
          for (const object of objects) {
            await s3Client.send(
              new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
            );
          }
        }
      }
      await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
    },
  };
}
```

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
```

```
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}
```

```
/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
```

```
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-
s3').S3Client}} param0
 */
export async function getObject({ bucket, s3Client }) {
  const prefix = "output/";
  const { Contents } = await s3Client.send(
    new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
  );

  if (!Contents.length) {
    throw new Error("No objects found in bucket.");
  }

  // Find the CSV file.
  const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

  if (!outputObject) {
    throw new Error(`No CSV file found in bucket with the prefix "${prefix}".`);
  }

  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: outputObject.Key,
    }),
  );

  return Body.transformToString();
}
```

この関数は、前述のライブラリ関数を使用して SageMaker 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 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

for JavaScript (v3) を使用した Secrets Manager SDK 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  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 詳細については、「 リファレンスGetSecretValue」の「」を参照してください。 AWS SDK for JavaScript API

for JavaScript (v3) SDK を使用した Amazon SESの例

次のコード例は、Amazon 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SES。

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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

アクション

`CreateReceiptFilter`

次のコード例は、`CreateReceiptFilter` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  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 詳細については、「 リファレンスCreateReceiptFilter」の「」を参照してください。 AWS SDK for JavaScript API

`CreateReceiptRule`

次の例は、`CreateReceiptRule` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスCreateReceiptRule」の「」を参照してください。 AWS SDK for JavaScript API

`CreateReceiptRuleSet`

次のコード例は、`CreateReceiptRuleSet` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスCreateReceiptRuleSet」の「」を参照してください。AWS SDK for JavaScript API

`CreateTemplate`

次の例は、`CreateTemplate` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスCreateTemplate」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteIdentity`

次の例は、DeleteIdentity を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteIdentity」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteReceiptFilter`

次の例は、`DeleteReceiptFilter` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteReceiptFilter」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteReceiptRule`

次の例は、`DeleteReceiptRule` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteReceiptRule」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteReceiptRuleSet`

次の例は、`DeleteReceiptRuleSet` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteReceiptRuleSet」の「」を参照してください。AWS SDK for JavaScript API

`DeleteTemplate`

次のコード例は、DeleteTemplate を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDeleteTemplate」の「」を参照してください。 AWS SDK for JavaScript API

`GetTemplate`

次の例は、`GetTemplate` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスGetTemplate」の「」を参照してください。 AWS SDK for JavaScript API

`ListIdentities`

次の例は、`ListIdentities` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスListIdentities」の「」を参照してください。 AWS SDK for JavaScript API

`ListReceiptFilters`

次のコード例は、`ListReceiptFilters` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスListReceiptFilters」の「」を参照してください。 AWS SDK for JavaScript API

`ListTemplates`

次のコード例は、`ListTemplates` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスListTemplates」の「」を参照してください。 AWS SDK for JavaScript API

`SendBulkTemplatedEmail`

次のコード例は、SendBulkTemplatedEmail を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスSendBulkTemplatedEmail」の「」を参照してください。 AWS SDK for JavaScript API

`SendEmail`

次のコード例は、SendEmail を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスSendEmail」の「」を参照してください。 AWS SDK for JavaScript API

SendRawEmail

次の例は、SendRawEmail を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスSendRawEmail」の「」を参照してください。 AWS SDK for JavaScript API

`SendTemplatedEmail`

次のコード例は、SendTemplatedEmail を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスSendTemplatedEmail」の「」を参照してください。AWS SDK for JavaScript API

`UpdateTemplate`

次のコード例は、UpdateTemplate を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスUpdateTemplate」の「」を参照してください。 AWS SDK for JavaScript API

`VerifyDomainIdentity`

次のコード例は、`VerifyDomainIdentity` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスVerifyDomainIdentity」の「」を参照してください。AWS SDK for JavaScript API

`VerifyEmailIdentity`

次の例は、`VerifyEmailIdentity` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 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 詳細については、「 リファレンスVerifyEmailIdentity」の「」を参照してください。 AWS SDK for JavaScript API

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 の JavaScript (v3)

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、文字起こし、翻訳し、Amazon Simple Email Service (Amazon) 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を示しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

DynamoDB データを追跡するウェブアプリケーションを作成する

次のコード例は、Amazon DynamoDB テーブル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 の JavaScript (v3)

Amazon DynamoDB を使用して、DynamoDB の作業データを追跡する動的ウェブアプリケーションAPI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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Amazon SES

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 の 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) 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SES。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 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 を使用して、フィルタリングされた作業項目の E メールレポートを送信しますSES。
- 付属の 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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Aurora
- Amazon RDS

- Amazon RDS Data Service
- Amazon SES

イメージPPE内の検出

次のコード例は、Amazon Rekognition を使用して画像内の個人用保護具 (PPE) を検出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 の JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Simple Storage Service (Amazon S3PPE) バケットにあるイメージ内の個人用保護具 () を検出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。Amazon S3 アプリケーションは、結果を Amazon DynamoDB テーブルに保存し、Amazon Simple Email Service (Amazon) 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SES。

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 PPEを使用して のイメージを分析します。 Amazon Rekognition
- Amazon の E メールアドレスを確認しますSES。
- 結果で DynamoDB テーブルを更新します。
- Amazon を使用して E メール通知を送信しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3
- 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 を使用してイメージ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 の 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。アプリは、Amazon Simple Email Service (Amazon) 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SES。

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 の E メールアドレスを確認しますSES。
- Amazon を使用して E メール通知を送信しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

for JavaScript (v3) SDK を使用した Amazon SNSの例

次のコード例は、Amazon 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SNS。

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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

開始方法

Amazon へようこそ SNS

次のコード例は、Amazon の使用を開始する方法を示していますSNS。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

SNS クライアントを初期化し、アカウント内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 詳細については、「 リファレンスListTopics」の「」を参照してください。 AWS SDK for JavaScript API

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

import { snsClient } from "../libs/snsClient.js";
```

```
export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

- 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、「 リファレンスCheckIfPhoneNumberIsOptedOut」の「」を参照してください。 AWS SDK for JavaScript API

`ConfirmSubscription`

次の例は、`ConfirmSubscription` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
```

```
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 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、「 リファレンスConfirmSubscription」の「」を参照してください。 AWS SDK for JavaScript API

`CreateTopic`

次の例は、`CreateTopic` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

/**
 * @param {string} topicName - The name of the topic to create.
```

```
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- 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、「 リファレンスCreateTopic」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteTopic`

次のコード例は、`DeleteTopic` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- 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、「 リファレンスDeleteTopic」の「」を参照してください。 AWS SDK for JavaScript API

GetSMSAttributes

次のコード例は、GetSMSAttributes を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
```

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

- 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、「 リファレンス」の「GetSMSAttributes」を参照してください。 AWS SDK for JavaScript API

GetTopicAttributes

次の例は、GetTopicAttributes を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

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

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

- 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、「 リファレンスGetTopicAttributes」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();

// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

- 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、「 リファレンスGetTopicAttributes」の「」を参照してください。 AWS SDK for JavaScript API

`ListSubscriptions`

次の例は、`ListSubscriptions` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- 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、「 リファレンスListSubscriptions」の「」を参照してください。 AWS SDK for JavaScript API

`ListTopics`

次の例は、`ListTopics` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
```

```
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- 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、「 リファレンスListTopics」の「」を参照してください。 AWS SDK for JavaScript API

`Publish`

次の例は、Publish を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

```
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                  if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

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, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
```

```
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
```

- 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、「 リファレンス」の「パブリッシュ」を参照してください。 AWS SDK for JavaScript API

`SetSMSAttributes`

次のコード例は、`SetSMSAttributes` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 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、「 リファレンス」の「SetSMSAttributes」を参照してください。 AWS SDK for JavaScript API

SetTopicAttributes

次の例は、SetTopicAttributes を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
```

```
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 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、「 リファレンスSetTopicAttributes」の「」を参照してください。 AWS SDK for JavaScript API

Subscribe

次の例は、Subscribe を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

```
/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

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;
};
```

フィルターを使用してトピックを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 詳細については、「 AWS SDK for JavaScript APIリファレンス」の「サブスクライブ」を参照してください。

`Unsubscribe`

次のコード例は、Unsubscribe を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 および クライアントモジュールをインポートし、 を呼び出しますAPI。

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

/**
```

```
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 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「サブスクリプション解除」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 の 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) 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ケーションを構築する方法を示しますSNS。

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、「」の詳細な例を参照してくださいGitHub。

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 の 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セットアップと実行の手順については、「」の詳細な例を参照してください GitHub。

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 ゲートウェイ
- 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 の 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。ストレージには Amazon Simple Storage Service (Amazon S3) を使用し、通知には Amazon Simple Notification Service (Amazon SQS) トピックにサブスクライブされている Amazon Simple Queue Service (Amazon SNS) 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と実行の手順については、「」の詳細な例を参照してくださいGitHub。

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 以外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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

これがこのワークフローのエントリーポイントです。

```
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";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

上記のコードにより必要な依存関係が提供され、ワークフローが開始されます。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.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
```

```
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
```

```
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }
```

```
async subscribeQueuesToTopic() {
  for (const [index, queue] of this.queues.entries()) {
    /**
     * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
     */
    const subscribeParams = {
      TopicArn: this.topicArn,
      Protocol: "sqs",
      Endpoint: queue.queueArn,
    };
    let tones = [];

    if (this.isFifo) {
      if (index === 0) {
        await this.logger.log(MESSAGES.fifoFilterNotice);
      }
      tones = await this.prompter.checkbox({
        message: MESSAGES.fifoFilterSelect.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
        choices: toneChoices,
      });

      if (tones.length) {
        subscribeParams.Attributes = {
          FilterPolicyScope: "MessageAttributes",
          FilterPolicy: JSON.stringify({
            tone: tones,
          }),
        };
      }
    }

    const { SubscriptionArn } = await this.snsClient.send(
      new SubscribeCommand(subscribeParams),
    );

    this.subscriptionArns.push(SubscriptionArn);

    await this.logger.log(
      MESSAGES.queueSubscribedNotice
        .replace("${QUEUE_NAME}", queue.queueName)
```

```
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
```

```
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
```

```
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }
```

```
  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- API 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
  - Publish
  - ReceiveMessage

- SetQueueAttributes
- Subscribe
- Unsubscribe

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信する 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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda でSNSイベントを消費する JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
```

```
}
```

を使用して Lambda でSNSイベントを消費する TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

for JavaScript (v3) SDK を使用した Amazon SQSの例

次のコード例は、Amazon 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SQS。

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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

開始方法

Amazon へようこそ SQS

次のコード例は、Amazon の使用を開始する方法を示していますSQS。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListQueues」の「」を参照してください。 AWS SDK for JavaScript API

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`

次のコード例は、ChangeMessageVisibility を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQS メッセージを受信し、タイムアウトの可視性を変更します。

```
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 詳細については、「 リファレンスChangeMessageVisibility」の「」を参照してください。AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQS メッセージを受信し、タイムアウトの可視性を変更します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- 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、「 リファレンスChangeMessageVisibility」の「」を参照してください。 AWS SDK for JavaScript API

`CreateQueue`

次のコード例は、`CreateQueue` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 Amazon SQS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 詳細については、「 リファレンスCreateQueue」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQS標準キューを作成します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
```

```
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

メッセージが到着するのを待機する Amazon SQSキューを作成します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 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、「 リファレンスCreateQueue」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteMessage`

次のコード例は、`DeleteMessage` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteMessage」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQS メッセージを受信および削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

- 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、「 リファレンスDeleteMessage」の「」を参照してください。 AWS SDK for JavaScript API

DeleteMessageBatch

次の例は、DeleteMessageBatch を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 {
  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 詳細については、「 リファレンスDeleteMessageBatch」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteQueue`

次のコード例は、DeleteQueue を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteQueue」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQSキューを削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスDeleteQueue」の「」を参照してください。 AWS SDK for JavaScript API

`GetQueueAttributes`

次のコード例は、`GetQueueAttributes` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスGetQueueAttributes」の「」を参照してください。 AWS SDK for JavaScript API

`GetQueueUrl`

次の例は、`GetQueueUrl` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQSキューURLの 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 詳細については、「 リファレンスGetQueueUrl」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQSキューURLの を取得します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 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、「 リファレンスGetQueueUrl」の「」を参照してください。 AWS SDK for JavaScript API

`ListQueues`

次の例は、`ListQueues` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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);
    urls.forEach((url) => 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 詳細については、「 リファレンスListQueues」の「」を参照してください。 AWS SDK for JavaScript API

SDK の JavaScript (v2)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQSキューを一覧表示します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- 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、「 リファレンスListQueues」の「」を参照してください。 AWS SDK for JavaScript API

`ReceiveMessage`

次の例は、`ReceiveMessage` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスReceiveMessage」の「」を参照してください。 AWS SDK for JavaScript API

SDK JavaScript (v2) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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、「 リファレンスReceiveMessage」の「」を参照してください。 AWS SDK for JavaScript API

SendMessage

次の例は、SendMessage を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスSendMessage」の「」を参照してください。 AWS SDK for JavaScript API

SDK JavaScript (v2) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

Amazon SQSキューにメッセージを送信します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
```

```
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- 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、「 リファレンスSendMessage」の「」を参照してください。 AWS SDK for JavaScript API

`SetQueueAttributes`

次のコード例は、`SetQueueAttributes` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスSetQueueAttributes」の「」を参照してください。 AWS SDK for JavaScript API

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 の 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) をストレージとして使用し、Amazon Simple Notification Service (Amazon SQS) トピックにサブスクライブされている Amazon Simple Queue Service (Amazon SNS) キューをポーリングする通知に使用します。

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、「」の詳細な例を参照してくださいGitHub。

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 以外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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

これがこのワークフローのエントリーポイントです。

```
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";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
```

```
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

上記のコードにより必要な依存関係が提供され、ワークフローが開始されます。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.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );
```

```
    const { Attributes } = await this.sqsClient.send(
      new GetQueueAttributesCommand({
        QueueUrl: response.QueueUrl,
        AttributeNames: ["QueueArn"],
      }),
    );

    this.queues.push({
      queueName,
      queueArn: Attributes.QueueArn,
      queueUrl: response.QueueUrl,
    });

    await this.logger.log(
      MESSAGES.queueCreatedNotice
        .replace("${QUEUE_NAME}", queueName)
        .replace("${QUEUE_URL}", response.QueueUrl)
        .replace("${QUEUE_ARN}", Attributes.QueueArn),
    );
  }
}

async attachQueueIamPolicies() {
  for (const [index, queue] of this.queues.entries()) {
    const policy = JSON.stringify(
      {
        Statement: [
          {
            Effect: "Allow",
            Principal: {
              Service: "sns.amazonaws.com",
            },
            Action: "sqs:SendMessage",
            Resource: queue.queueArn,
            Condition: {
              ArnEquals: {
                "aws:SourceArn": this.topicArn,
              },
            },
          },
        ],
      },
      null,
```

```
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
```

```
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
```

```
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
```

```
              },
            },
          }
        : {}),
    }),
  );

  const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
  });

  if (publishAnother) {
    await this.publishMessages();
  }
}

async receiveAndDeleteMessages() {
  for (const queue of this.queues) {
    const { Messages } = await this.sqsClient.send(
      new ReceiveMessageCommand({
        QueueUrl: queue.queueUrl,
      }),
    );

    if (Messages) {
      await this.logger.log(
        MESSAGES.messagesReceivedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
      console.log(Messages);

      await this.sqsClient.send(
        new DeleteMessageBatchCommand({
          QueueUrl: queue.queueUrl,
          Entries: Messages.map((message) => ({
            Id: message.MessageId,
            ReceiptHandle: message.ReceiptHandle,
          })),
        }),
      );
    } else {
      await this.logger.log(
```

```
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
```

```
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- API 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
  - Publish
  - 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

次のコード例は、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信する Lambda 関数を実装する方法を示していますSQS。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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda でSQSイベントを消費する JavaScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

を使用して Lambda でSQSイベントを消費する TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Amazon SQSトリガーを使用した Lambda 関数のバッチアイテム失敗のレポート

次のコード例は、 SQSキュー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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。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。

を使用して Lambda でSQSバッチアイテムの失敗をレポートする JavaScript。

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

を使用して Lambda でSQSバッチアイテムの失敗をレポートする TypeScript。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
```

```
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

JavaScript (v3) SDK に を使用する 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";
```

```
/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
  return response;
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- API 詳細については、「 リファレンスStartExecution」の「」を参照してください。 AWS SDK for JavaScript API

AWS STS JavaScript (v3) SDK に を使用する 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

アクション

`AssumeRole`

次のコード例は、`AssumeRole` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 のロールを引き受けます。

```
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 詳細については、「 リファレンスAssumeRole」の「」を参照してください。 AWS SDK for JavaScript API

SDK JavaScript (v2) 用の

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

```
// Load the AWS SDK for Node.js
const AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

var roleToAssume = {
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",
  RoleSessionName: "session1",
  DurationSeconds: 900,
```

```
};
var roleCreds;

// Create the STS service object
var sts = new AWS.STS({ apiVersion: "2011-06-15" });

//Assume Role
sts.assumeRole(roleToAssume, function (err, data) {
  if (err) console.log(err, err.stack);
  else {
    roleCreds = {
      accessKeyId: data.Credentials.AccessKeyId,
      secretAccessKey: data.Credentials.SecretAccessKey,
      sessionToken: data.Credentials.SessionToken,
    };
    stsGetCallerIdentity(roleCreds);
  }
});

//Get Arn of current identity
function stsGetCallerIdentity(creds) {
  var stsParams = { credentials: creds };
  // Create STS service object
  var sts = new AWS.STS(stsParams);

  sts.getCallerIdentity({}, function (err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data.Arn);
    }
  });
}
```

- API 詳細については、「 リファレンスAssumeRole」の「」を参照してください。 AWS SDK for JavaScript API

AWS Support JavaScript (v3) SDK に を使用する 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 Support。

「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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

開始方法

こんにち AWS Supportは

次のコード例は、 AWS Supportの使用を開始する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

`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.",
```

```
      );
    } else {
      throw err;
    }
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- API 詳細については、「 リファレンスDescribeServices」の「」を参照してください。 AWS SDK for JavaScript API

トピック

- 基礎
- アクション

基礎

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 の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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.",
      );
```

```
    } else {
      throw err;
    }
  }
};

/**
 * Select a service from the list returned from DescribeServices.
 */
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const selectedService = await inquirer.select({
    message:
      "Select a service. Your support case will be created for this service. The
 list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
};

/**
 * @param {{ categories: import('@aws-sdk/client-support').Category[]}} service
 */
export const getCategory = async (service) => {
  const selectedCategory = await inquirer.select({
    message: "Select a category.",
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const selectedSeverityLevel = await inquirer.select({
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

/**
 * Create a new support case
```

```
 * @param {{
 *  selectedService: import('@aws-sdk/client-support').Service
 *  selectedCategory: import('@aws-sdk/client-support').Category
 *  selectedSeverityLevel: import('@aws-sdk/client-support').SeverityLevel
 * }} selections
 * @returns
 */
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodaysOpenCases = async () => {
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });

  const { cases } = await client.send(command);

  if (cases.length === 0) {
    throw new Error(
      "Unexpected number of cases. Expected more than 0 open cases.",
    );
  }
  return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
```

```
  const command = new AddAttachmentsToSetCommand({
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
  });
  const { communications } = await client.send(command);
  return communications;
};

/**
 * @param {import('@aws-sdk/client-support').Communication[]} communications
 */
export const getFirstAttachment = (communications) => {
  const firstCommWithAttachment = communications.find(
    (c) => c.attachmentSet.length > 0,
  );
  return firstCommWithAttachment?.attachmentSet[0].attachmentId;
};

// Get an attachment.
export const getAttachment = async (attachmentId) => {
  const command = new DescribeAttachmentCommand({
    attachmentId,
```

```
  });
  const { attachment } = await client.send(command);
  return attachment;
};

// Resolve the case matching the given case ID.
export const resolveCase = async (caseId) => {
  const shouldResolve = await inquirer.confirm({
    message: `Do you want to resolve ${caseId}?`,
  });

  if (shouldResolve) {
    const command = new ResolveCaseCommand({
      caseId: caseId,
    });

    await client.send(command);
    return true;
  }
  return false;
};

/**
 * Find a specific case in the list of provided cases by case ID.
 * If the case is not found, and the results are paginated, continue
 * paging through the results.
 * @param {{
 *   caseId: string,
 *   cases: import('@aws-sdk/client-support').CaseDetails[]
 *   nextToken: string
 * }} options
 * @returns
 */
export const findCase = async ({ caseId, cases, nextToken }) => {
  const foundCase = cases.find((c) => c.caseId === caseId);

  if (foundCase) {
    return foundCase;
  }

  if (nextToken) {
    const response = await client.send(
      new DescribeCasesCommand({
        nextToken,
```

```
        includeResolvedCases: true,
      }),
    );
    return findCase({
      caseId,
      cases: response.cases,
      nextToken: response.nextToken,
    });
  }

  throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
 select one.
    const selectedCategory = await getCategory(selectedService);

    // Provide the severity available severity levels for the account and prompt the
 user to select one.
```

```
  const selectedSeverityLevel = await getSeverityLevel();

  // Create a support case.
  console.log("\nCreating a support case.");
  caseId = await createCase({
    selectedService,
    selectedCategory,
    selectedSeverityLevel,
  });
  console.log(`Support case created: ${caseId}`);

  // Display a list of open support cases created today.
  const todaysOpenCases = await retry(
    { intervalInMs: 1000, maxRetries: 15 },
    getTodaysOpenCases,
  );
  console.log(
    `\nOpen support cases created today: ${todaysOpenCases.length}`,
  );
  console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

  // Create an attachment set.
  console.log("\nCreating an attachment set.");
  const attachmentSetId = await createAttachmentSet();
  console.log(`Attachment set created: ${attachmentSetId}`);

  // Add the attachment set to the support case.
  console.log(`\nAdding attachment set to ${caseId}`);
  await linkAttachmentSetToCase(attachmentSetId, caseId);
  console.log(`Attachment set added to ${caseId}`);

  // List the communications for a support case.
  console.log(`\nListing communications for ${caseId}`);
  const communications = await getCommunications(caseId);
  console.log(
    communications
      .map(
        (c) =>
          `Communication created on ${c.timeCreated}. Has
${c.attachmentSet.length} attachments.`,
      )
      .join("\n"),
  );
```

```
    // Describe the first attachment.
    console.log(`\nDescribing attachment ${attachmentSetId}`);
    const attachmentId = getFirstAttachment(communications);
    const attachment = await getAttachment(attachmentId);
    console.log(
      `Attachment is the file '${
        attachment.fileName
      }' with data: \n${new TextDecoder().decode(attachment.data)}`,
    );

    // Confirm that the support case should be resolved.
    const isResolved = await resolveCase(caseId);
    if (isResolved) {
      // List the resolved cases and include the one previously created.
      // Resolved cases can take a while to appear.
      console.log(
        "\nWaiting for case status to be marked as resolved. This can take some
 time.",
      );
      const resolvedCases = await retry(
        { intervalInMs: 20000, maxRetries: 15 },
        () => getTodaysResolvedCases(caseId),
      );
      console.log("Resolved cases:");
      console.log(resolvedCases.map((c) => c.caseId).join("\n"));
    }
  } catch (err) {
    console.error(err);
  }
};
```

- API 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`

次のコード例は、AddAttachmentsToSet を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスAddAttachmentsToSet」の「」を参照してください。 AWS SDK for JavaScript API

`AddCommunicationToCase`

次のコード例は、AddCommunicationToCase を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスAddCommunicationToCase」の「」を参照してください。 AWS SDK for JavaScript API

`CreateCase`

次のコード例は、CreateCase を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスCreateCase」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeAttachment`

次の例は、DescribeAttachment を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeAttachment」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeCases`

次の例は、`DescribeCases` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeCases」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeCommunications`

次の例は、`DescribeCommunications` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeCommunications」の「」を参照してください。 AWS SDK for JavaScript API

`DescribeSeverityLevels`

次のコード例は、`DescribeSeverityLevels` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスDescribeSeverityLevels」の「」を参照してください。 AWS SDK for JavaScript API

`ResolveCase`

次の例は、`ResolveCase` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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 { 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 詳細については、「 リファレンスResolveCase」の「」を参照してください。 AWS SDK for JavaScript API

JavaScript (v3) SDKに を使用する 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

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 の 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) をストレージとして使用し、Amazon Simple Notification Service (Amazon SQS) トピックにサブスクライブされている Amazon Simple Queue Service (Amazon SNS) キューをポーリングする通知に使用します。

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、「」の詳細な例を参照してくださいGitHub。

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 の 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、「」の「 GitHub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

for JavaScript (v3) を使用した Amazon Transcribe SDK 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

トピック

アクション

`DeleteMedicalTranscriptionJob`

次のコード例は、`DeleteMedicalTranscriptionJob` を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteMedicalTranscriptionJob」の「」を参照してください。 AWS SDK for JavaScript API

`DeleteTranscriptionJob`

次の例は、`DeleteTranscriptionJob` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスDeleteTranscriptionJob」の「」を参照してください。 AWS SDK for JavaScript API

`ListMedicalTranscriptionJobs`

次の例は、`ListMedicalTranscriptionJobs` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListMedicalTranscriptionJobs」の「」を参照してください。 AWS SDK for JavaScript API

ListTranscriptionJobs

次のコード例は、ListTranscriptionJobs を使用する方法を示しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスListTranscriptionJobs」の「」を参照してください。 AWS SDK for JavaScript API

`StartMedicalTranscriptionJob`

次の例は、StartMedicalTranscriptionJob を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスStartMedicalTranscriptionJob」の「」を参照してください。 AWS SDK for JavaScript API

`StartTranscriptionJob`

次の例は、`StartTranscriptionJob` を使用する方法を説明しています。

SDK の JavaScript (v3)

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、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。

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 詳細については、「 リファレンスStartTranscriptionJob」の「」を参照してください。 AWS SDK for JavaScript API

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 の JavaScript (v3)

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、文字起こし、翻訳し、Amazon Simple Email Service (Amazon) 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を示しますSES。

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、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

for JavaScript (v3) を使用した Amazon Translate SDK 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 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います。このリンクには、コンテキスト内でコードをセットアップして実行する方法の手順が記載されています。

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 の JavaScript (v3)

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、文字起こし、翻訳し、Amazon Simple Email Service (Amazon) 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を示しますSES。

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、「」の詳細な例を参照してくださいGitHub。

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 の JavaScript (v3)

Amazon Lex を使用してウェブアプリケーション内に Chatbot APIを作成し、ウェブサイトの訪問者をエンゲージさせる方法を示します。

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、 AWS SDK for JavaScript デ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 の 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、「」の「 GitHub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 とユーザーの間で共有される責任です。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。当社のセキュリティ責任は で最優先事項であり 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ています。

クラウドにおけるセキュリティ — お客様の責任は、使用している AWS サービス、およびデータの機密性、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 、 AWS コンプライアンスプログラム による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におけるデータ保護
- 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バージョンを適用する

この AWS 製品またはサービスにおけるデータ保護

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。お客様は、このインフラストラクチャでホストされてい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 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FAQ」を参照してください。欧州におけるデータ保護の詳細については、AWS 「 セキュリティブログ」のAWS 「責任共有モデル」とGDPR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 () 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IAM。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。1TLS.2 が必要で、1.3 TLS をお勧めします。
- を使用して 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。
- AWS 暗号化ソリューションと、 内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 サービス。
- Amazon Macie などの高度なマネージド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 AWS を介して にアクセスするときに FIPS 140-3 検証済みの暗号化モジュールが必要な場合はAPI、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は、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配置しないことを強くお勧めします。これは、コンソール、、または を使用して、この AWS 製品またはサービスまたは他の AWS サービス を使用する場合API AWS CLIも同様です AWS SDKs。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。URL を外部サーバーに提供する場合は、そのサーバーへのリクエストを検証URLするために認証情報を 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 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 リソースの使用を承認する (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 AWS サービス 使用できる です。

トピック

- 対象者

- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- の AWS サービス 仕組み IAM
- AWS ID とアクセスのトラブルシューティング

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 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与えられます。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ことがあります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。の機能にアクセスできない場合は AWS、AWS ID とアクセスのトラブルシューティング「」または AWS 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。

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。次に、サービスユーザーのアクセス許可を変更するリクエストをIAM管理者に送信する必要があります。このページの情報を確認して、 の基本概念を理解してくださいIAM。会社IAMで を使用する方法の詳細については AWS、使用している の AWS サービス ユーザーガイドを参照してください。

IAM 管理者 - IAM管理者は、 へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります AWS。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 IAM、 AWS 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 の認証情報は、フェデレーティッド 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と、管理者は以前に IAMロールを使用して ID フェデレーションをセッ

トアップしていました。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、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」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に暗号で署名します。 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。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、「 IAMユーザーガイド」のAWS API「リクエストの署名」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。例えば、 AWS 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「 ユーザーガイド」の「 での多要素認証 (MFA) AWS IAM の使用」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 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「ルートユーザーの認証情報を必要とするタスク」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、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS サービス 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 ソースを通じて提供された認証情報 AWS 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 ID がにアクセスすると 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。Identity Center でユーザーとグループを作成することも、独自の IAM ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。IAM Identity Center の詳細については、「 ユーザーガイド」のIAM「Identity Center とはAWS IAM Identity Center 」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 内のアイデンティティです。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持つ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ユーザーとの長期的な認証情報を必要とする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、「 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でアクセスキーを定期的にローテーションするIAM」を参照してください。

IAM グループは、IAMユーザーのコレクション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、 という名前のグループIAMAdminsを作成し、そのグループにIAMリソースを管理するアクセス許可を付与できます。

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、「 ユーザーガイド」のIAM「 (ロールの代わりに) ユーザーを作成する場合IAM」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 内のアイデンティティです。これは IAM ユーザーと似ていますが、特定のユーザーに関連付けられていません。IAM ロール を切り替える AWS Management Console ことで、 で ロールを一時的に引き受けることができます。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 を使用しますURL。ロールの使用方法の詳細については、「 ユーザーガイド」のIAM「ロールの使用IAM」を参照してください。

IAM 一時的な認証情報を持つ ロールは、以下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の詳細については、「 ユーザーガイド」の「サードパーティー ID プロバイダーのロールの作成IAM」を参照してください。IAM Identity Center を使用する場合は、アクセス許可セットを設定します。ID が認証後にアクセスできる内容を制御するために、IAMIdentity Center はアクセス許可セットを のロールに関連付けますIAM。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的なIAMユーザーアクセス許可 – IAM ユーザーまたはロールは、 IAMロールを引き受けて、特定のタスクに対して異なるアクセス許可を一時的に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、別のアカウントのユーザー (信頼されたプリンシパル) がアカウントのリソースにアクセスすることを許可できます。クロスアカウントアクセスを許可する主な方法は、ロールを使用することです。ただし、一部の では AWS サービス、(ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。クロスアカウントアクセスのロールとリソースベースのポリシーの違いについては、「 ユーザーガイド」の「 でのクロスアカウントリソースアクセスIAMIAM」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 サービス を使用します AWS サービス。例えば、サービスで呼び出しを行うと、そのサービスが Amazon でアプリケーションを実行 EC2したり、Amazon S3 にオブジェクトを保存したりするのが一般的です。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 で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 サービス、ダウンストリームサービス AWS サービス へのリクエストのリクエスト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 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権限が必要です。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 – サービスロールは、ユーザーに代わってアクションを実行するためにサービスが引き受ける IAMロールです。IAM 管理者は、 内からサービスロールを作成、変更、削除できますIAM。詳細については、「 ユーザーガイド」の「 にアクセス許可を委任するロールの作成 AWS サービスIAM」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 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、サービスにリンクされた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 で実行されているアプリケーション EC2 – 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インスタンス内にアクセスキーを保存するよりも望ましいです。 AWS ロールをEC2インスタンスに割り当て、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 ロールが含まれており、EC2インスタンスで実行されているプログラムが一時的な認証情報を取得できるようにします。詳細については、「 ユーザーガイド」の「 IAMロールを使用して Amazon EC2インスタンスで実行されているアプリケーションにアクセス許可を付与するIAM」を参照してください。

IAM ロールとIAMユーザーのどちらを使用するかについては、「 ユーザーガイド」の「 (ユーザーではなく) IAMロールを作成する場合IAM」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、 AWS ID 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、「 ユーザーガイド」のJSON「ポリシーの概要IAM」を参照してください。

管理者はポリシーを使用して AWS JSON、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。必要なリソースに対してアクションを実行するアクセス許可をユーザーに付与するために、IAM管理者はIAMポリシーを作成できます。その後、管理者はIAMポリシーをロールに追加し、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に使用する方法に関係なく、アクションのアクセス許可を定義します。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 からロール情報を取得できますAPI。

アイデンティティベースのポリシー

ID ベースのポリシーは、IAMユーザー、ユーザーのグループ、ロールなどの ID にアタッチできるJSONアクセス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のポリシーを作成する方法については、「 ユーザーガイド」のIAM「ポリシーの作成IAM」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。管理ポリシーとインラインポリシーのどちら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選択」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、リソースにアタッチする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の例としては、IAMロールの信頼ポリシー や Amazon S3 バケットポリシー などがあります。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コントロールできます。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 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IAMでは、 の AWS 管理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 (ACLs)

アクセスコントロールリスト (ACLs) は、リソースへのアクセス許可を持つプリンシパル (アカウントメンバー、ユーザー、またはロール) を制御します。ACLs 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を使用しません。

Amazon S3、 AWS WAF、および Amazon VPCは、 をサポートするサービスの例ですACLs。の詳細についてはACLs、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、一般的ではない追加のポリシータイプをサポートします。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のポリシーがIAMエンティティ (IAMユーザーまたはロール) に付与できるアクセス許可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) に対する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、 AWS アカウント ビジネスが所有する複数の 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 (SCPs) をアカウントの一部またはすべてに適用できます。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SCP を制限します AWS アカウントのルートユーザー。Organizations と の詳細についてはSCPs、「 AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- 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、「 ユーザーガイド」の「セッションポリシーIAM」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。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、「 ユーザーガイド」の「ポリシー評価ロジックIAM」を参照してください。

の AWS サービス 仕組み IAM

がほとんどの IAM 機能と AWS サービス 連携する方法の概要を把握するには、 IAMユーザーガイドのAWS 「 と連携する のサービスIAM」を参照してください。

で特定の を使用する方法については AWS サービス IAM、関連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 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IAM。

トピック

- でアクションを実行する権限がない AWS
- iam を実行する権限がありません。PassRole
- 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ない AWS

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、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。

次の例のエラーは、mateojacksonIAMユーザーが コンソールを使用して架空の*my-example-widget*リソースの詳細を表示しようとしているが、架空のawes:*GetWidget*アクセス許可がない場合に発生します。

```
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 を実行する権限がありません。PassRole

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 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。

次の例のエラーは、 というIAMユーザーがコンソールを使用して marymajor でアクションを実行しようする場合に発生します AWS。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。サインイン認証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 (ACLs) 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、ユーザーにリソースへのアクセスを許可できます。

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の AWS サービス 仕組み IAM。
- 所有している のリソースへのアクセスを提供する方法については、 AWS アカウント 「 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IAM」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「外部認証されたユーザーへのアクセスの提供 (ID フェデレーション)」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いについては、「 ユーザーガイド」の「 でのクロスアカウントリソースアクセスIAMIAM」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 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、コンプライアンスプログラムAWS サービス による対象範囲内のコンプライアンスプログラムを参照し、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「コンプライアンスプログラム」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、「 でのレポートのダウンロード AWS Artifact」の」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 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 を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、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に重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスHIPAAのセキュリティとコンプライアンスのためのアーキテクチャ – このホワイトペーパーでは、企業が AWS を使用して HIPAA対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべての AWS サービス がHIPAA対象となるわけではありません。詳細については、HIPAA「対象サービスリファレンス」を参照してください。

- 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、ガイダンスを保護し AWS サービス 、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に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、リソース設定が社内プラクティス、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 サービス 、 内のセキュリティ状態を包括的に確認できます 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リストについては、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 – これにより AWS アカウント、疑わしい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監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS サービス を検出します。 GuardDuty は、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DSS、 PCI などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS 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 、 AWS コンプライアンスプログラム による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、物理的に分離および分離された複数の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 、 AWS コンプライアンスプログラム による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保護されています AWS 。 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。

が AWS 公開したAPI呼び出しを使用して、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) 。1TLS.2 が必要で、1.3 TLS をお勧めします。
- (Ephemeral Diffie-HellmanPFS) や DHE (Elliptic Curve Ephemeral Diffie-Hellman) などの完全前方秘匿性 ECDHE () を備えた暗号スイート。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。

さらに、 リクエストは、 IAMプリンシパルに関連付けられたアクセスキー ID と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 、 AWS コンプライアンスプログラム による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バージョンを適用する

AWS サービスとの通信時にセキュリティを強化するには、1.2 TLS 以降を使用する AWS SDK for JavaScript ように を設定します。

> **Important**
>
> AWS SDK for JavaScript v3 は、特定の AWS サービスエンドポイントでサポートされている最上位TLSバージョンを自動的にネゴシエートします。オプションで、1.2 や 1.3 TLS など、アプリケーションで必要な最小TLSバージョンを適用できますが、1.3 TLS は一部の AWS サービスエンドポイントではサポートされていないため、1.3 TLS を適用すると一部の呼び出しが失敗する場合があることに注意してください。

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。

Node.js TLSでの検証と強制

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、1.3 をサポートする OpenSSL 1.1.1b TLS の最小バージョンを使用します。 AWS SDK for JavaScript v3 のデフォルトは、使用可能な場合は TLS 1.3 を使用しますが、必要に応じて下位バージョンに設定されます。

OpenSSL と のバージョンを確認する TLS

コンピュータで Node.js が使用する Open SSL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'
```

コンピュータで Node.js がTLS使用する のバージョンを取得するには、Node シェルを起動し、次のコマンドを順番に実行します。

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

次の例に示すように、最後のコマンドは TLS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 を使用しTLS、呼び出しが成功しなかった場合TLSは別のバージョンの のネゴシエートを試みます。

の最小バージョンを強制する TLS

Node.js は、呼び出しが失敗TLSすると 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 コード内のリクエストごとに、このネゴシエーション中に最小許容TLSバージョンを適用できます。

コマンドラインから最小TLSバージョンを指定するには、Node.js バージョン 11.0.0 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

1TLS.2 が最小許容バージョンであることを強制するには、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の最小許容TLSバージョンを指定するには、次の例に示すように、 `http0ptions`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(
            {
                secureProtocol: 'TLSv1_2_method'
            }
        )
    })
});
```

Enforce TLS 1.3

TLS 1.3 が最小許容バージョンであることを適用するには、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の最小許容TLSバージョンを指定するには、次の例に示すように、 `httpOptions`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(
            {
                secureProtocol: 'TLSv1_3_method'
            }
        )
    })
});
```

ブラウザスクリプトTLSでの検証と強制

ブラウザスクリプト JavaScript で SDKの を使用する場合、ブラウザの設定によって、使用されるのバージョンTLSが制御されます。ブラウザTLSで使用される のバージョンは、スクリプトで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TLSで使用されるのバージョンを検証して適用するには、特定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. セキュリティカテゴリまで下にスクロールし、「1.2 を使用するTLS」のオプション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。*Internet options*.
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. ユーザー 1.2 TLS の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. セキュリティカテゴリまで下にスクロールし、「1.2 を使用するTLS」のオプション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. 1.2 のプロトコルをデフォルトにするには、整数値を TLS 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、1.2 TLS は自動的に有効になります。

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用の 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 JavaScript。

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 に相当するものは、AWS SDK for JavaScript GitHub リポジトリ の移行ガイドで確認できます。

AWS SDK for JavaScript v3 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

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 を使用するように移行するaws-sdk-js-codemodのに役立ちますAPIs。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

(v3) SDK用 のバージョン JavaScript 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 で記述されており TypeScript、静的型付けなど多くの利点があります。

> **Important**
>
> このガイドの v3 のコード例は、6 () ECMAScript で記述されていますES6。ES6 では、コードをより最新で読みやすくし、より多くのことを行うための新しい構文と新機能が導入され

ています。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

バージョン 2 SDK for JavaScript (v2) では AWS SDK、次のように 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SDKが多くの AWS サービスを使用している場合、 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簡単なコード例は、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、コールバックパターンまたは async/await パターンを使用して v2 コマンドをパッケージ内で直接実行します。

v3 コマンドの使用

v3 は、各 AWS サービスパッケージに一連のコマンドを提供し、その AWS サービスのオペレーションを実行できるようにします。 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 の で v2 コマンドを使用するには JavaScript、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

の v2 SDK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段階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));
```

AWS SDK for JavaScript v2 と v3 の違い

このセクションでは、 AWS SDK for JavaScript v2 から 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、すぐに参考になります。

- のモジュラーパッケージ AWS SDK for JavaScript
- モジュラー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 に相当する 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ー
- 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
- ウェイターと署名者
- 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ー

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 の該当するコマンド (など) はPutObject PutBucketCors、リクエストペイロードの 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が変換されるかどうか。
  - v3: を廃止しました。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が歪んでいるために失敗したクロックスキュー修正および再試行リクエストを適用するかどうか。
  - v3: を廃止しました。SDK は常にクロックスキュー補正を適用します。
- systemClockOffset
  - v2: すべての署名時間に適用され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。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、関数は再び呼び出されます。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: を廃止しました。SDK 必要に応じて、 は常にホスト名プレフィックスを挿入します。

- `httpOptions`

  低レベル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、 の v3 APIリファレンスを参照してください NodeHttpHandler。

  すべての v3 リクエストHTTPSはデフォルトで を使用します。カスタム を提供するだけで済みますhttpsAgent。

```
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、 を指定する必要がありますhttpAgent。

```
const { Agent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");

const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: new Agent({
            /*params*/
        }),
    }),
    endpoint: "http://example.com",
});
```

  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、 の v3 APIリファレンスを参照してください FetchHttpHandler。

```
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URLをプロキシするための 。
  - v3: エージェントを使用してプロキシをセットアップするには、「Node.js のプロキシの設定」に従います。
- 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: を廃止しました。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: 使用できません。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: を廃止しました。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。の詳細については、「 の v3 APIリファレンス RetryInputConfig」を参照してください。は にmaxAttemptsする必要があります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必要があるかどうか。
  - v3: を廃止しました。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の送信先のリージョン。
  - v3: 変更なし。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ための一連のオプション。
  - v3: を廃止しました。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をアドレス指定しているかどうか (ルートAPIエンドポイントをアドレス指定している場合は失敗) 。
  - v3: を に変更しましたbucketEndpoint。の詳細については、「 の v3 APIリファレンス bucketEndpoint」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 の使用時に 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URLsのパススタイルを強制するかどうか。
  - v3: の名前を に変更しましたforcePathStyle。
- s3UseArnRegion

  - v2: リクエストリージョンを、リクエストされたリソースの から推測されたリージョンで上書きするかどうかARN。
  - 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: を廃止しました。region 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: を廃止しました。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設定を上書き) 。
  - v3: を廃止しました。vV2サポートされている署名 V2 SDKは によって非推奨になりましたAWS。v3 は署名 v4 のみをサポートしています。
- sslEnabled
  - v2: リクエストに対して SSL が有効になっているかどうか。
  - 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: を廃止しました。STS クライアントは、特定のリージョンに設定されている場合、常にリージョンエンドポイントを使用します。リージョンを に設定aws-globalして、STS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。
- useAccelerateEndpoint
  - v2: S3 サービスで Accelerate エンドポイントを使用するかどうか。
  - v3: 変更なし。

認証情報プロバイダ

v2 では、 SDK for は、選択でき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 JavaScript を提供し、最も一般的なすべてのプロバイダーから 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の SDK for による AWS 認証情報の JavaScript 解決方法です。

- v2: Node.js CredentialProviderChainでは、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、チェーンは有効な認証情報が解決されるまで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、認証情報を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むため、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。

元の TemporaryCredentialsは v2 ChainableTemporaryCredentialsで を優先して廃止されました。

- 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 ID 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: @aws/credential-providersパッケージには 2 Cognito Identity Credential Provider つの認証情報プロバイダー関数があり、1 つは 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 を再呼び出ししません GetId。

  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「Simplified Flow」を実装します。の呼び出しcognito:GetOpenIdTokenと呼び出しを含む「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: を廃止しました。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: は使用できません。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 (デフォルトは ~/.aws/credentialsまたは AWS_SHARED_CREDENTIALS_FILE環境変数で定義されます) 。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 ファイルと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 でよく使用されます。

- 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 },
  }),
});
```

Web Identity 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 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

間違ったリージョンが 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 (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 では、ソケットを解放して接続を新しいトラフィックに開放し続けるために、ストリームまたはガベージを消費してクライアントまたはそのリクエストハンドラーを収集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 型で DynamoDB APIs を呼び出すことができます。したがって、属性値の概念を抽象化することで、Amazon DynamoDB の項目の操作を簡素化できます。
- 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 クライアントを使用する点が異なります。

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、デベロッパーは 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
    };
  })
);
```

パッケージ README では、その他の例と設定を使用できます。

ウェイターと署名者

このページでは、 AWS SDK for JavaScript v3 でのウェイ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、すべてのウェイ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イターの入力で指定する必要があります。例えば、新しく作成されたバケットの準備が整うのを待つwaitFor("bucketExists")には、 を呼び出す必要があります。

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、必要な特定の目的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小さくし、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はすべて、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へのアクセス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、 で 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 呼び出しの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送信のチェックサムを計算し、取得し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して持つ 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するカスタムを提供できました。

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ることはありません。

```
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フィールドで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",
});
```

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 をサブスクライブできます。

変更	説明	日付
発表	Internet Explorer 11 の end-of-support リマインダーで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。およびツールリファレンスガイドへの AWS SDKs認識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	1.3 TLS に関する情報を追加しました。	2022 年 3 月 31 日
AWS Lambda チュートリアルの更新	Amazon DynamoDBの表にデータを送信するブラウザベースのアプリケーションを構築する方法を示すチュートリアルを追加しました。	2020 年 10 月 20 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	AWS SDK for JavaScript v3 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 に必要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

Amazon DynamoDBのサービス例が更新されました	AWS SDK for JavaScript V3 の Amazon DynamoDB 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