Guide du développeur pour la version 3 du SDK

AWS SDK pour JavaScript

AWS SDK pour JavaScript: Guide du développeur pour la version 3 du SDK

Table of Contents

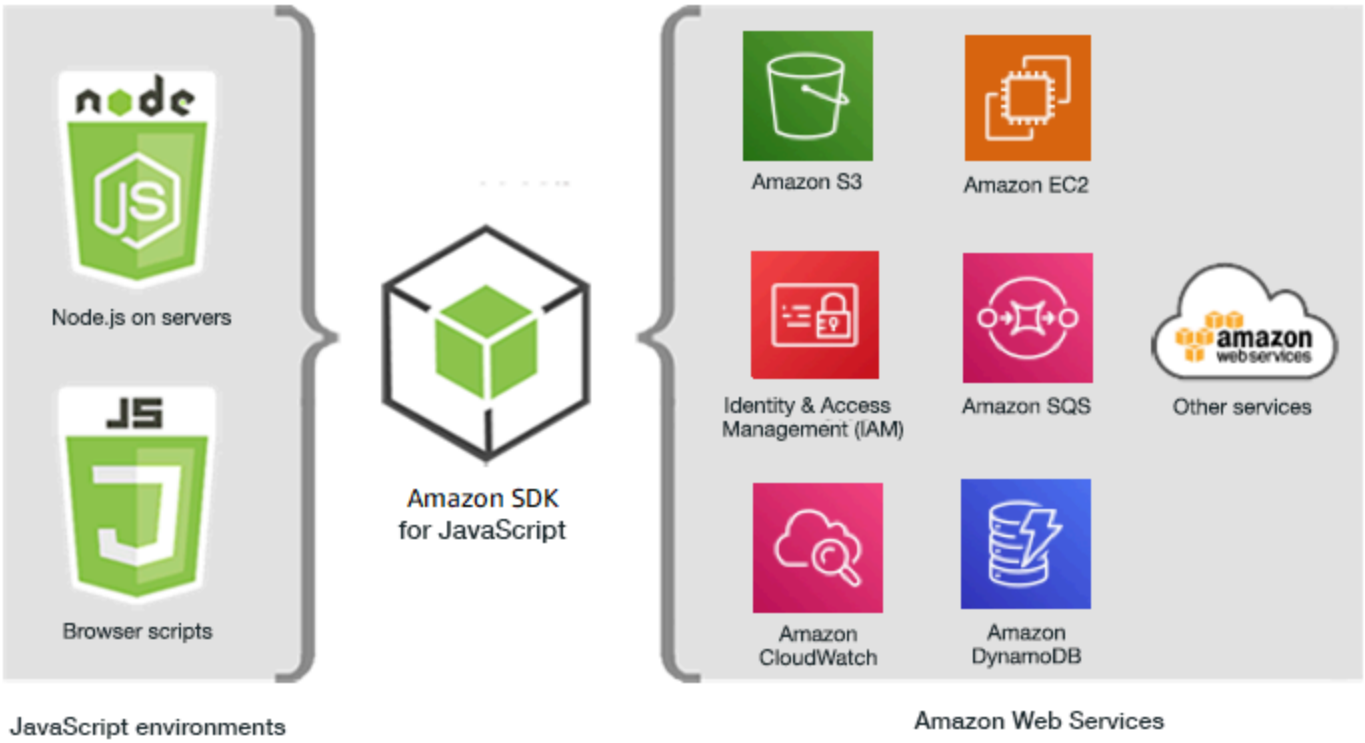
Le guide de référence de l'API AWS SDK pour JavaScript V3 décrit en détail toutes les opérations de l'API pour la AWS SDK pour JavaScript version 3 (V3).

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.

Qu'est-ce que c'est AWS SDK pour JavaScript ?

Bienvenue dans le guide du AWS SDK pour JavaScript développeur. Ce guide fournit des informations générales sur l'installation et la configuration du AWS SDK pour JavaScript. Il vous présente également des exemples et un didacticiel d'exécution de divers AWS services à l'aide du AWS SDK pour JavaScript.

Le guide de référence de l'API AWS SDK pour JavaScript v3 fournit une JavaScript API pour les AWS services. Vous pouvez utiliser l' JavaScript API pour créer des bibliothèques ou des applications pour Node.js ou le navigateur.



Commencez avec le SDK

Si vous êtes prêt à vous familiariser avec le SDK, suivez les exemples présentés surMise en route.

Pour configurer votre environnement de développement, consultezConfigurez le SDK pour JavaScript.

Si vous utilisez actuellement la version 2.x du SDK pour JavaScript, consultez Migrer vers la version 3 pour obtenir des conseils spécifiques.

Si vous recherchez des exemples de code pour Services AWS, consultezSDK pour exemples de JavaScript code (v3).

Maintenance et prise en charge des versions majeures du SDK

Pour plus d'informations sur la maintenance et le support des versions majeures du SDK et de leurs dépendances sous-jacentes, consultez les informations suivantes dans le guide de référence AWS SDKs and Tools :

- AWS SDKs et politique de maintenance des outils
- AWS SDKs et matrice de support des versions d'outils

Utilisation du kit SDK avec Node.js

Node.js est un environnement d'exécution multiplateforme permettant d'exécuter des applications côté serveur JavaScript . Vous pouvez configurer Node.js sur une instance Amazon Elastic Compute Cloud (Amazon EC2) pour qu'il s'exécute sur un serveur. Vous pouvez également utiliser Node.js pour écrire des AWS Lambda fonctions à la demande.

L'utilisation du SDK pour Node.js est différente de la manière dont vous l'utilisez JavaScript dans un navigateur Web. La différence provient de la façon dont vous chargez le kit SDK et dont vous récupérez les informations d'identification nécessaires à l'accès aux services web spécifiques. Lorsque l'utilisation d'un fichier particulier APIs diffère entre Node.js et le navigateur, nous signalons ces différences.

Utilisation du SDK avec AWS Amplify

Pour les applications Web, mobiles et hybrides basées sur un navigateur, vous pouvez également utiliser la AWS Amplify bibliothèque sur. GitHub Il étend le SDK pour JavaScript, en fournissant une interface déclarative.

> **Note**
>
> Les frameworks tels qu'Amplify peuvent ne pas offrir le même support de navigateur que le SDK pour. JavaScript Consultez la documentation du framework pour plus de détails.

Utilisation du SDK avec des navigateurs Web

Tous les principaux navigateurs Web prennent en charge l'exécution de JavaScript. JavaScript le code qui s'exécute dans un navigateur Web est souvent appelé côté client JavaScript.

Pour obtenir la liste des navigateurs pris en charge par le AWS SDK pour JavaScript, voirNavigateurs Web pris en charge.

L'utilisation du SDK pour JavaScript un navigateur Web est différente de la façon dont vous l'utilisez pour Node.js. La différence provient de la façon dont vous chargez le kit SDK et dont vous récupérez les informations d'identification nécessaires à l'accès aux services web spécifiques. Lorsque l'utilisation d'un fichier particulier APIs diffère entre Node.js et le navigateur, nous signalons ces différences.

Utilisation des navigateurs dans la V3

La version 3 vous permet de regrouper et d'inclure dans le navigateur uniquement le SDK pour les JavaScript fichiers dont vous avez besoin, ce qui réduit les frais généraux.

Pour utiliser la version 3 du SDK JavaScript dans vos pages HTML, vous devez regrouper les modules client requis et toutes les JavaScript fonctions requises dans un seul JavaScript fichier à l'aide de Webpack, et l'ajouter dans une balise `<head>` de script dans vos pages HTML. Par exemple :

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

> **Note**
>
> Pour plus d'informations sur Webpack, consultezRegroupez des applications avec Webpack.

Pour utiliser la version 2 du SDK pour JavaScript, vous devez plutôt ajouter une balise de script pointant vers la dernière version du SDK V2. Pour plus d'informations, consultez l'exemple dans le Guide du AWS SDK pour JavaScript développeur v2.

Cas d’utilisation courants

L'utilisation du SDK pour les scripts JavaScript intégrés au navigateur permet de réaliser un certain nombre de cas d'utilisation convaincants. Voici quelques idées de choses que vous pouvez intégrer

dans une application de navigateur en utilisant le SDK pour accéder JavaScript à divers services Web.

- Créez une console personnalisée pour les AWS services dans laquelle vous pouvez accéder aux fonctionnalités de différentes régions et services et les combiner afin de répondre au mieux aux besoins de votre organisation ou de votre projet.
- Utilisez Amazon Cognito Identity pour permettre aux utilisateurs authentifiés d'accéder aux applications et aux sites Web de votre navigateur, notamment en utilisant l'authentification par un tiers via Facebook et d'autres.
- Utilisez Amazon Kinesis pour traiter les flux de clics ou d'autres données marketing en temps réel.
- Utilisez Amazon DynamoDB pour la persistance des données sans serveur, telles que les préférences individuelles des utilisateurs pour les visiteurs du site Web ou les utilisateurs d'applications.
- AWS Lambda Utilisez-le pour encapsuler une logique propriétaire que vous pouvez invoquer à partir de scripts de navigateur sans télécharger ni révéler votre propriété intellectuelle aux utilisateurs.

À propos des exemples

Vous pouvez parcourir le SDK pour trouver des JavaScript exemples dans le référentiel d'exemples de AWS code.

Ressources

Outre ce guide, les ressources en ligne suivantes sont disponibles pour le SDK pour JavaScript les développeurs :

- AWS SDK pour JavaScript Guide de référence de l'API V3
- AWS SDKs et Guide de référence des outils : contient des paramètres, des fonctionnalités et d'autres concepts fondamentaux courants entre AWS SDKs.
- JavaScript Blog du développeur
- AWS Re : Publier
- JavaScript exemples dans la bibliothèque AWS de code
- AWS Référentiel d'exemples de code
- Chaîne Gitter

- Stack Overflow
- Questions relatives à Stack Overflow taggeDAWS -sdk-js
- GitHub
  - Source du SDK
  - Source de documentation

Commencez avec le AWS SDK pour JavaScript

Le AWS SDK pour JavaScript permet d'accéder aux services Web dans un navigateur ou dans un environnement Node.js. Cette section contient des exercices de démarrage qui vous montrent comment utiliser le AWS SDK JavaScript dans chacun de ces JavaScript environnements.

Rubriques

Authentification du SDK avec AWS

Vous devez définir la manière dont votre code s'authentifie AWS lorsque vous développez avec Services AWS. Vous pouvez configurer l'accès programmatique aux AWS ressources de différentes manières en fonction de l'environnement et de l' AWS accès dont vous disposez.

Pour choisir votre méthode d'authentification et la configurer pour le SDK, consultez la section Authentification et accès dans le guide de référence des outils AWS SDKs et.

Nous recommandons aux nouveaux utilisateurs qui se développent localement et qui n'ont pas reçu de méthode d'authentification de la part de leur employeur de les configurer AWS IAM Identity Center. Cette méthode inclut l'installation AWS CLI pour faciliter la configuration et pour vous connecter régulièrement au portail AWS d'accès. Si vous choisissez cette méthode, votre environnement doit contenir les éléments suivants une fois que vous avez terminé la procédure d'authentification IAM Identity Center décrite dans le guide de référence AWS SDKs and Tools :

- Le AWS CLI, que vous utilisez pour démarrer une session de portail d' AWS accès avant d'exécuter votre application.
- `AWSconfig`Fichier partagé doté d'un `[default]` profil avec un ensemble de valeurs de configuration pouvant être référencées à partir du SDK. Pour connaître l'emplacement de ce fichier, reportez-vous à la section Emplacement des fichiers partagés dans le Guide de référence des outils AWS SDKs et.

- Le `config` fichier partagé définit le `region`paramètre. Cela définit la valeur par défaut Région AWS que le SDK utilise pour les AWS demandes. Cette région est utilisée pour les demandes de service du SDK qui ne sont pas spécifiées avec une région à utiliser.
- Le SDK utilise la configuration du fournisseur de jetons SSO du profil pour obtenir des informations d'identification avant d'envoyer des demandes à. AWS La `sso_role_name` valeur, qui est un rôle IAM connecté à un ensemble d'autorisations IAM Identity Center, permet d'accéder à l'utilisateur Services AWS dans votre application.

  Le `config` fichier d'exemple suivant montre un profil par défaut configuré avec la configuration du fournisseur de jetons SSO. Le `sso_session` paramètre du profil fait référence à la `sso-session`section nommée. La `sso-session` section contient les paramètres permettant de lancer une session sur le portail AWS d'accès.

```
[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
```

Le AWS SDK pour la JavaScript version 3 n'a pas besoin de packages supplémentaires (tels que `SSO` et`SSOOIDC`) à ajouter à votre application pour utiliser l'authentification IAM Identity Center.

Pour plus de détails sur l'utilisation explicite de ce fournisseur d'informations d'identification, consultez `fromSSO()`le site Web npm (gestionnaire de packages Node.js).

Démarrer une session sur le portail AWS d'accès

Avant d'exécuter une application qui y accède Services AWS, vous avez besoin d'une session de portail d' AWS accès active pour que le SDK utilise l'authentification IAM Identity Center pour résoudre les informations d'identification. En fonction de la durée de session que vous avez configurée, votre accès finira par expirer et le SDK rencontrera une erreur d'authentification. Pour vous connecter au portail AWS d'accès, exécutez la commande suivante dans le AWS CLI.

```
aws sso login
```

Si vous avez suivi les instructions et que vous avez configuré un profil par défaut, il n'est pas nécessaire d'appeler la commande avec une `--profile` option. Si la configuration de votre fournisseur de jetons SSO utilise un profil nommé, la commande est `aws sso login --profile named-profile`.

Pour éventuellement vérifier si vous avez déjà une session active, exécutez la AWS CLI commande suivante.

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

Si votre session est active, la réponse à cette commande indique le compte IAM Identity Center et l'ensemble d'autorisations configurés dans le `config` fichier partagé.

> **Note**
>
> Si vous disposez déjà d'une session active sur le portail AWS d'accès et que vous l'exécutez`aws sso login`, il ne vous sera pas demandé de fournir d'informations d'identification.
> Le processus de connexion peut vous demander d'autoriser l' AWS CLI accès à vos données. Comme AWS CLI il repose sur le SDK pour Python, les messages d'autorisation peuvent contenir des variantes du `botocore` nom.

Utilisation des informations de connexion à la console

Vous pouvez utiliser vos identifiants de connexion existants à la console de AWS gestion pour accéder aux AWS services par programmation. Après un flux d'authentification basé sur un navigateur, AWS génère des informations d'identification temporaires qui fonctionnent avec les outils de développement locaux tels que la AWS CLI et le AWS SDK pour. JavaScript Cette fonctionnalité simplifie le processus de configuration et de gestion des informations d'identification de la AWS CLI. Pour savoir comment démarrer, suivez les instructions de connexion pour le développement AWS local à l'aide des informations d'identification de la console.

Lorsque vous exécutez la aws login commande, vous pouvez effectuer une sélection parmi vos sessions de console actives ou vous connecter via le flux d'authentification basé sur le navigateur, ce qui générera automatiquement des informations d'identification temporaires. Le AWS SDK actualise

JavaScript automatiquement les informations d'identification 5 minutes avant leur expiration, chaque ensemble d'informations d'identification étant valide pendant 12 heures au maximum. Pour de plus amples informations, veuillez consulter `fromLoginCredentials()`.

Plus d'informations d'authentification

Les utilisateurs humains, également connus sous le nom identités humaines, sont les personnes, les administrateurs, les développeurs, les opérateurs et les consommateurs de vos applications. Ils doivent disposer d'une identité pour accéder à vos AWS environnements et applications. Les utilisateurs humains membres de votre organisation, c'est-à-dire vous, le développeur, sont appelés identités du personnel.

Utilisez des informations d'identification temporaires lors de l'accès AWS. Vous pouvez utiliser un fournisseur d'identité pour vos utilisateurs humains afin de fournir un accès fédéré aux AWS comptes en assumant des rôles fournissant des informations d'identification temporaires. Pour une gestion centralisée des accès, nous vous recommandons d'utiliser AWS IAM Identity Center (IAM Identity Center) pour gérer l'accès à vos comptes et les autorisations associées à ces comptes. Pour d'autres alternatives, consultez les rubriques suivantes :

- Pour en savoir plus sur les bonnes pratiques, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.
- Pour créer des AWS informations d'identification à court terme, consultez la section Informations d'identification de sécurité temporaires dans le guide de l'utilisateur IAM.
- Pour en savoir plus sur les autres fournisseurs de AWS SDK pour les fournisseurs d'informations d'identification JavaScript V3, consultez la section Fournisseurs d'informations d'identification standardisés dans le guide de référence AWS SDKs and Tools.

Commencez avec Node.js

Ce guide explique comment initialiser un package NPM, ajouter un client de service à votre package et utiliser le JavaScript SDK pour appeler une action de service.

Le scénario

Créez un nouveau package NPM avec un fichier principal qui effectue les opérations suivantes :

- Crée un bucket Amazon Simple Storage Service
- Place un objet dans le compartiment Amazon S3

- Lit l'objet dans le compartiment Amazon S3
- Confirme si l'utilisateur souhaite supprimer des ressources

Conditions préalables

Avant de pouvoir exécuter cet exemple, vous devez effectuer les opérations suivantes :

- Configurez l'authentification de votre SDK. Pour de plus amples informations, veuillez consulter Authentification du SDK avec AWS.
- Installez Node.js. AWS recommande d'utiliser la version Active LTS de Node.js pour le développement.

Étape 1 : Configuration de la structure des packages et installation des packages clients

Pour configurer la structure des packages et installer les packages clients :

1. Créez un nouveau dossier `nodegetstarted` pour contenir le package.
2. À partir de la ligne de commande, accédez au nouveau dossier.
3. Exécutez la commande suivante pour créer un `package.json` fichier par défaut :

```
npm init -y
```

4. Exécutez la commande suivante pour installer le package client Amazon S3 :

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

5. Ajoutez `"type": "module"` au `package.json` fichier. Cela indique à Node.js d'utiliser la syntaxe ESM moderne. La finale `package.json` devrait ressembler à ce qui suit :

```
{
  "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"
}
```

Étape 2 : ajouter les importations et le code SDK nécessaires

Ajoutez le code suivant à un fichier nommé `index.js` dans le `nodegetstarted` dossier.

```
// This is used for getting user input.
import { createInterface } from "node:readline/promises";

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

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

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

```
    Bucket: bucketName,
  }),
);

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

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

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

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

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

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

```
            new DeleteObjectCommand({ Bucket: bucketName, Key: object.Key }),
          );
        }
      }
    }

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

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

L'exemple de code se trouve ici GitHub.

Étape 3 : Exécutez l'exemple

> **Note**
>
> N'oubliez pas de vous connecter ! Si vous utilisez IAM Identity Center pour vous authentifier, n'oubliez pas de vous connecter à l'aide de la AWS CLI `aws sso login` commande.

1. Exécutez `node index.js`.
2. Choisissez de vider et de supprimer le compartiment.
3. Si vous ne supprimez pas le compartiment, veillez à le vider manuellement et à le supprimer ultérieurement.

Commencez dans le navigateur

Cette section présente un exemple qui montre comment exécuter la version 3 (V3) du AWS SDK JavaScript dans le navigateur.

> **Note**
>
> L'exécution de la V3 dans le navigateur est légèrement différente de la version 2 (V2). Pour de plus amples informations, veuillez consulter Utilisation des navigateurs dans la V3.

Pour d'autres exemples d'utilisation (V3) du AWS SDK pour JavaScript, voir. SDK pour exemples de JavaScript code (v3)

Cet exemple d'application Web vous montre :

- Comment accéder aux AWS services à l'aide d'Amazon Cognito pour l'authentification.
- Comment lire une liste d'objets dans un compartiment Amazon Simple Storage Service (Amazon S3) à l'aide d' Gestion des identités et des accès AWS un rôle (IAM).

> **Note**
>
> Cet exemple n'est pas utilisé AWS IAM Identity Center pour l'authentification.

Scénario

Amazon S3 est un service de stockage d'objets qui offre une évolutivité, une disponibilité des données, une sécurité et des performances de pointe. Vous pouvez utiliser Amazon S3 pour stocker des données sous forme d'objets dans des conteneurs appelés buckets. Pour plus d'informations sur Amazon S3, consultez le guide de l'utilisateur Amazon S3.

Cet exemple vous montre comment configurer et exécuter une application Web qui assume un rôle IAM pour lire depuis un compartiment Amazon S3. L'exemple utilise la bibliothèque frontale React et les outils frontaux Vite pour fournir un JavaScript environnement de développement. L'application Web utilise un pool d'identités Amazon Cognito pour fournir les informations d'identification nécessaires pour accéder AWS aux services. L'exemple de code inclus illustre les modèles de base pour le chargement et l'utilisation du AWS SDK pour JavaScript les applications Web.

Étape 1 : créer un pool d'identités et un rôle IAM Amazon Cognito

Dans cet exercice, vous allez créer et utiliser un pool d'identités Amazon Cognito afin de fournir un accès non authentifié à votre application Web pour le service Amazon S3. La création d'un pool

d'identités crée également un rôle Gestion des identités et des accès AWS (IAM) pour prendre en charge les utilisateurs invités non authentifiés. Dans cet exemple, nous ne travaillerons qu'avec le rôle d'utilisateur non authentifié afin de nous concentrer sur la tâche. Vous pouvez vous former à la prise en charge d'un fournisseur d'identité et des utilisateurs authentifiés ultérieurement. Pour plus d'informations sur l'ajout d'un pool d'identités Amazon Cognito, consultez Tutoriel : Création d'un pool d'identités dans le guide du développeur Amazon Cognito.

Pour créer un pool d'identités Amazon Cognito et le rôle IAM associé

1. Connectez-vous à la console Amazon Cognito AWS Management Console et ouvrez-la à l'adresse. https://console.aws.amazon.com/cognito/
2. Dans le volet de navigation de gauche, choisissez Identity pools.
3. Choisissez Créer un groupe d'identités.
4. Dans Configurer la confiance du pool d'identités, choisissez Accès invité pour l'authentification des utilisateurs.
5. Dans Configurer les autorisations, choisissez Créer un nouveau rôle IAM et entrez un nom (par exemple, getStartedRole) dans le nom du rôle IAM.
6. Dans Configurer les propriétés, entrez un nom (par exemple, getStartedPool) dans Nom du pool d'identités.
7. Dans Vérifier et créer, confirmez les sélections que vous avez effectuées pour votre nouvelle réserve d'identités. Sélectionnez Modifier pour revenir dans l'assistant et modifier des paramètres. Lorsque vous avez terminé, sélectionnez Créer un groupe d'identités.
8. Notez l'ID du pool d'identités et la région du pool d'identités Amazon Cognito récemment créé. Vous avez besoin de ces valeurs pour les remplacer *IDENTITY_POOL_ID* et *REGION* les insérerÉtape 4 : configurer le code du navigateur.

Après avoir créé votre pool d'identités Amazon Cognito, vous êtes prêt à ajouter les autorisations nécessaires à votre application Web pour Amazon S3.

Étape 2 : ajouter une politique au rôle IAM créé

Pour activer l'accès à un compartiment Amazon S3 dans votre application Web, utilisez le rôle IAM non authentifié (par exemple getStartedRole) créé pour votre pool d'identités Amazon Cognito (par exemple,). getStartedPool Vous devez pour cela associer une politique IAM au rôle. Pour plus d'informations sur la modification des rôles IAM, consultez la section Modification d'une politique d'autorisations de rôle dans le Guide de l'utilisateur IAM.

Pour ajouter une politique Amazon S3 au rôle IAM associé aux utilisateurs non authentifiés

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à https://console.aws.amazon.com/iam/l'adresse.
2. Dans le volet de navigation de gauche, choisissez Rôles.
3. Choisissez le nom du rôle que vous souhaitez modifier (par exemple, getStartedRole), puis cliquez sur l'onglet Autorisations.
4. Choisissez Ajouter des autorisations, puis Attacher des politiques.
5. Sur la page Ajouter des autorisations pour ce rôle, recherchez puis cochez la case pour AmazonS3 ReadOnlyAccess.

   > **Note**
   >
   > Vous pouvez utiliser ce processus pour autoriser l'accès à n'importe quel AWS service.

6. Choisissez Ajouter des autorisations.

Après avoir créé votre pool d'identités Amazon Cognito et ajouté des autorisations pour Amazon S3 à votre rôle IAM pour les utilisateurs non authentifiés, vous êtes prêt à ajouter et à configurer un compartiment Amazon S3.

Étape 3 : ajouter un compartiment et un objet Amazon S3

Au cours de cette étape, vous allez ajouter un compartiment et un objet Amazon S3 pour l'exemple. Vous allez également activer le partage de ressources entre origines (CORS) pour le bucket. Pour plus d'informations sur la création de buckets et d'objets Amazon S3, consultez Getting started with Amazon S3 dans le guide de l'utilisateur Amazon S3.

Pour ajouter un compartiment et un objet Amazon S3 avec CORS

1. Connectez-vous à la console Amazon S3 AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/s3/.
2. Dans le volet de navigation de gauche, choisissez Buckets, puis Create bucket.
3. Entrez un nom de compartiment conforme aux règles de dénomination des compartiments (par exemple, getstartedbucket) et choisissez Create bucket.
4. Choisissez le bucket que vous avez créé, puis cliquez sur l'onglet Objets. Choisissez ensuite Upload.

5. Sous Fichiers et dossiers, choisissez Ajouter des fichiers.
6. Choisissez un fichier à charger, puis choisissez Ouvrir. Choisissez ensuite Upload pour terminer le téléchargement de l'objet dans votre bucket.
7. Choisissez ensuite l'onglet Autorisations de votre compartiment, puis sélectionnez Modifier dans la section Partage de ressources entre origines (CORS). Entrez le code JSON suivant :

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

8. Sélectionnez Enregistrer les modifications.

Après avoir ajouté un compartiment Amazon S3 et un objet, vous êtes prêt à configurer le code du navigateur.

Étape 4 : configurer le code du navigateur

L'exemple d'application consiste en une application React d'une seule page. Les fichiers de cet exemple se trouvent ici GitHub.

Pour configurer l'exemple d'application

1. Installez Node.js.
2. À partir de la ligne de commande, clonez le référentiel d'exemples de AWS code :

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

3. Accédez à l'exemple d'application :

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

4. Exécutez la commande suivante pour installer les packages requis :

```
npm install
```

5. Ouvrez ensuite `src/App.tsx` dans un éditeur de texte et effectuez les opérations suivantes :

   - *YOUR_IDENTITY_POOL_ID*Remplacez-le par l'ID du pool d'identités Amazon Cognito que vous avez indiqué. Étape 1 : créer un pool d'identités et un rôle IAM Amazon Cognito
   - Remplacez la valeur de région par la région attribuée à votre compartiment Amazon S3 et à votre pool d'identités Amazon Cognito. Notez que les régions des deux services doivent être identiques (par exemple, us-east-2).
   - *bucket-name*Remplacez-le par le nom du bucket dans lequel vous l'avez crééÉtape 3 : ajouter un compartiment et un objet Amazon S3.

Après avoir remplacé le texte, enregistrez le `App.tsx` fichier. Vous êtes maintenant prêt à exécuter l'application Web.

Étape 5 : Exécuter l'exemple

Pour exécuter l'exemple d'application

1. À partir de la ligne de commande, accédez à l'exemple d'application :

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

2. À partir de la ligne de commande, exécutez la commande suivante :

```
npm run dev
```

   L'environnement de développement Vite s'exécutera avec le message suivant :

```
VITE v4.3.9  ready in 280 ms

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

3. Dans votre navigateur Web, accédez à l'URL ci-dessus (par exemple, http://localhost:5173). L'exemple d'application vous montrera une liste de noms de fichiers d'objets dans votre compartiment Amazon S3.

Nettoyage

Pour nettoyer les ressources que vous avez créées au cours de ce didacticiel, procédez comme suit :

- Dans la console Amazon S3, supprimez tous les objets et tous les compartiments créés (par exemple, getstartedbucket).
- Dans la console IAM, supprimez le nom du rôle (par exemple, getStartedRole).
- Dans la console Amazon Cognito, supprimez le nom du pool d'identités (par exemple, getStartedPool).

Commencer à utiliser React Native

Ce didacticiel vous montre comment créer une application React Native à l'aide de la CLI React Native.

Ce didacticiel vous montre :

- Comment installer et inclure le AWS SDK pour les modules de JavaScript la version 3 (V3) utilisés par votre projet.
- Comment écrire du code qui se connecte à Amazon Simple Storage Service (Amazon S3) afin de créer et de supprimer un compartiment Amazon S3.

Scénario

Amazon S3 est un service cloud qui vous permet de stocker et de récupérer n'importe quel volume de données à tout moment, où que vous soyez sur le Web. React Native est un framework de développement qui permet de créer des applications mobiles. Ce didacticiel explique comment créer

une application React Native qui se connecte à Amazon S3 pour créer et supprimer un compartiment Amazon S3.

L'application utilise le AWS SDK suivant pour JavaScript APIs :

- Constructeur `CognitoIdentityClient`
- Constructeur `S3`

Tâches préalables

> **Note**
>
> Si vous avez déjà effectué l'une des étapes suivantes via d'autres didacticiels ou une configuration existante, ignorez ces étapes.

Cette section fournit la configuration minimale nécessaire pour effectuer ce didacticiel. Vous ne devez pas considérer cela comme une configuration complète. Pour cela, consultez Configurez le SDK pour JavaScript.

- Installez les outils suivants :
  - npm
  - Node.js
  - Xcode si vous testez sur iOS
  - Android Studio si vous testez sur Android
- Configurez votre environnement de développement React Native
- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez le AWS SDK requis pour les modules tiers JavaScript et les modules tiers. Suivez les instructions figurant sur GitHub.
- Vous devez définir le mode d'authentification de votre code AWS lorsque vous développez avec des AWS services. Pour de plus amples informations, veuillez consulter Authentification du SDK avec AWS.

> **Note**
>
> Dans cet exemple, le rôle IAM doit être défini pour utiliser les autorisations Amazon S3 FullAccess.

Étape 1 : créer un pool d'identités Amazon Cognito

Dans cet exercice, vous allez créer et utiliser un pool d'identités Amazon Cognito afin de fournir un accès non authentifié à votre application pour le service Amazon S3. La création d'un pool d'identités crée également deux rôles Gestion des identités et des accès AWS (IAM), l'un pour prendre en charge les utilisateurs authentifiés par un fournisseur d'identité et l'autre pour prendre en charge les utilisateurs invités non authentifiés.

Dans cet exercice, nous allons uniquement avoir recours à un rôle utilisateur non authentifié afin de rester concentré sur la tâche. Vous pouvez vous former à la prise en charge d'un fournisseur d'identité et des utilisateurs authentifiés ultérieurement.

Pour créer un pool d'identités Amazon Cognito

1. Connectez-vous à la console Amazon Cognito AWS Management Console et ouvrez-la sur la console Amazon Web Services.
2. Choisissez Identity Pools sur la page d'ouverture de la console.
3. Sur la page suivante, choisissez Créer un groupe d'identités.

   > **Note**
   >
   > S'il n'existe aucun autre pool d'identités, la console Amazon Cognito ignore cette page et ouvre la page suivante à la place.

4. Dans Configurer la confiance du pool d'identités, choisissez Accès invité pour l'authentification des utilisateurs.
5. Dans Configurer les autorisations, choisissez Créer un nouveau rôle IAM et entrez un nom (par exemple, getStartedReactRôle) dans le nom du rôle IAM.
6. Dans Configurer les propriétés, entrez un nom (par exemple, getStartedReactPool) dans Nom du pool d'identités.

7. Dans Vérifier et créer, confirmez les sélections que vous avez effectuées pour votre nouvelle réserve d'identités. Sélectionnez Modifier pour revenir dans l'assistant et modifier des paramètres. Lorsque vous avez terminé, sélectionnez Créer un groupe d'identités.
8. Notez l'ID du pool d'identités et la région de ce groupe d'identités nouvellement créé. Vous avez besoin de ces valeurs pour remplacer *`region`* et *`identityPoolId`* dans le script de votre navigateur.

Après avoir créé votre pool d'identités Amazon Cognito, vous êtes prêt à ajouter les autorisations nécessaires à votre application React Native pour Amazon S3.

Étape 2 : ajouter une politique au rôle IAM créé

Pour permettre l'accès par script de navigateur à Amazon S3 afin de créer et de supprimer un compartiment Amazon S3, utilisez le rôle IAM non authentifié créé pour votre pool d'identités Amazon Cognito. Cela nécessite que vous ajoutiez une politique IAM au rôle. Pour plus d'informations sur les rôles IAM, consultez la section Création d'un rôle pour déléguer des autorisations à un AWS service dans le Guide de l'utilisateur IAM.

Pour ajouter une politique Amazon S3 au rôle IAM associé aux utilisateurs non authentifiés

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à https://console.aws.amazon.com/iam/l'adresse.
2. Dans le volet de navigation de gauche, choisissez Rôles.
3. Choisissez le nom du rôle que vous souhaitez modifier (par exemple, getStartedRole), puis cliquez sur l'onglet Autorisations.
4. Choisissez Ajouter des autorisations, puis Attacher des politiques.
5. Sur la page Ajouter des autorisations pour ce rôle, recherchez puis cochez la case pour AmazonS3 ReadOnlyAccess.

   > **Note**
   >
   > Vous pouvez utiliser ce processus pour autoriser l'accès à n'importe quel AWS service.

6. Choisissez Ajouter des autorisations.

Après avoir créé votre pool d'identités Amazon Cognito et ajouté des autorisations pour Amazon S3 à votre rôle IAM pour les utilisateurs non authentifiés, vous êtes prêt à créer l'application.

Étape 3 : créer une application à l'aide de create-react-native-app

Créez une application React Native en exécutant la commande suivante.

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

Étape 4 : Installation du package Amazon S3 et des autres dépendances

Dans le répertoire du projet, exécutez les commandes suivantes pour installer le package Amazon S3.

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

Cette commande installe le package Amazon S3 dans votre projet et est mise à jour `package.json` pour répertorier Amazon S3 en tant que dépendance du projet. Vous pouvez trouver des informations sur ce package en recherchant "@aws -sdk » sur le site Web de https://www.npmjs.com/ npm.

Ces packages et le code associé sont installés dans le sous-répertoire `node_modules` de votre projet.

Pour plus d'informations sur l'installation des packages Node.js, voir Téléchargement et installation de packages localement et Création de modules Node.js sur le site Web npm (Node.js package manager). Pour plus d'informations sur le téléchargement et l'installation du AWS SDK pour JavaScript, consultezInstallez le SDK pour JavaScript.

Installez les autres dépendances requises pour l'authentification.

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

Étape 5 : Écrire le code React Native

Ajoutez le code suivant auApp`.tsx`. Remplacez *`identityPoolId`* et *`region`* par l'ID du pool d'identités et la région dans lesquels votre compartiment Amazon S3 sera créé.

```
import React, { useCallback, useState } from "react";
import { Button, StyleSheet, Text, TextInput, View } from "react-native";
import "react-native-get-random-values";
import "react-native-url-polyfill/auto";
```

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

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

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

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

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

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

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

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

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

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

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

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

export default App;
```

Le code importe d'abord les dépendances React, React Native et AWS SDK requises.

Dans la fonction App :

- L'objet S3Client est créé, en spécifiant les informations d'identification à l'aide du pool d'identités Amazon Cognito créé précédemment.
- Les méthodes `createBucket` et `deleteBucket` créer et supprimer le compartiment spécifié, respectivement.
- La vue React Native affiche un champ de saisie de texte permettant à l'utilisateur de spécifier un nom de compartiment Amazon S3, ainsi que des boutons pour créer et supprimer le compartiment Amazon S3 spécifié.

La JavaScript page complète est disponible ici GitHub.

Étape 6 : Exécuter l'exemple

> **Note**
>
> N'oubliez pas de vous connecter ! Si vous utilisez IAM Identity Center pour vous authentifier, n'oubliez pas de vous connecter à l'aide de la AWS CLI `aws sso login` commande.

Pour exécuter l'exemple `webios`, exécutez ou `android` commandez à l'aide de npm.

Voici un exemple de sortie d'exécution d'une `ios` commande sur macOS.

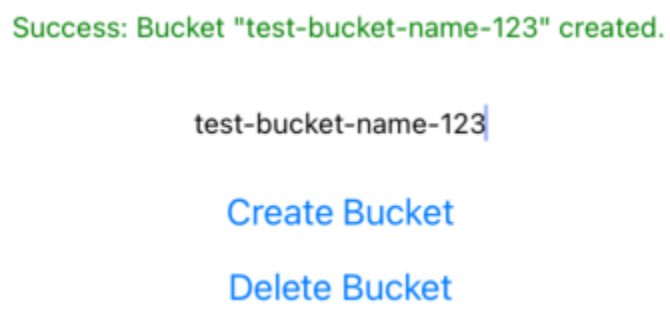
```
$ npm run ios

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

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

success Successfully launched the app on the simulator
```

Voici un exemple de sortie d'exécution d'une `android` commande sur macOS.

```
$ npm run android

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

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

```
info Installing the app...

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

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

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

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

Entrez le nom du bucket que vous souhaitez créer ou supprimer et cliquez sur Create Bucket ou Delete Bucket. La commande correspondante sera envoyée à Amazon S3 et un message de réussite ou d'erreur s'affichera.



Améliorations possibles

Voici des variantes de cette application que vous pouvez utiliser pour explorer plus en détail l'utilisation du AWS SDK JavaScript dans une application React Native.

- Ajoutez un bouton pour répertorier les compartiments Amazon S3 et fournissez un bouton de suppression à côté de chaque compartiment répertorié.
- Ajoutez un bouton pour placer un objet texte dans un compartiment.
- Intégrez un fournisseur d'identité externe tel que Facebook ou Amazon à utiliser avec le rôle IAM authentifié.

Configurez le SDK pour JavaScript

Les rubriques de cette section expliquent comment installer et charger le SDK pour JavaScript que vous puissiez accéder aux services Web pris en charge par le SDK.

Rubriques

Conditions préalables

Installez Node.js. AWS recommande d'utiliser la version Active LTS de Node.js pour le développement.

Rubriques

Configuration d'un environnement AWS Node.js

Pour configurer un environnement AWS Node.js dans lequel vous pouvez exécuter votre application, appliquez l'une des méthodes suivantes :

- Choisissez une Amazon Machine Image (AMI) sur laquelle Node.js est préinstallé. Créez ensuite une EC2 instance Amazon à l'aide de cette AMI. Lors de la création de votre EC2 instance Amazon, choisissez votre AMI dans le AWS Marketplace. AWS Marketplace Recherchez le fichier Node.js et choisissez une option AMI qui inclut une version préinstallée de Node.js (32 bits ou 64 bits).
- Créez une EC2 instance Amazon et installez-y Node.js. Pour plus d'informations sur l'installation de Node.js sur une instance Amazon Linux, consultezConfiguration de Node.js sur une EC2 instance Amazon.
- Créez un environnement sans serveur en utilisant AWS Lambda pour exécuter Node.js en tant que fonction Lambda. Pour plus d'informations sur l'utilisation de Node.js dans une fonction Lambda, voir Modèle de programmation (Node.js) dans le Guide du AWS Lambda développeur.

- Déployez votre application Node.js sur AWS Elastic Beanstalk. Pour plus d'informations sur l'utilisation de Node.js avec Elastic Beanstalk, consultez la section Déploiement d' AWS Elastic Beanstalk applications Node.js dans le manuel du développeur.AWS Elastic Beanstalk

Navigateurs Web pris en charge

Il AWS SDK pour JavaScript prend en charge tous les navigateurs Web modernes.

Dans la version 3.567.0 ou ultérieure, le SDK pour JavaScript émet ES2 021 artefacts, qui prennent en charge les versions minimales suivantes.

Navigateur	Version
Google Chrome	85,0 et plus
Mozilla Firefox	80,0 et plus
Opera	71 ans et plus
Microsoft Edge	85,0 et plus
Apple Safari	14,1+
Samsung Internet	14,0 et plus

Dans les versions 3.183.0 à 3.566.0, le SDK JavaScript utilise ES2 020 artefacts, qui prennent en charge les versions minimales suivantes.

Navigateur	Version
Google Chrome	80,0 et plus
Mozilla Firefox	80,0 et plus
Opera	63,0 et plus
Microsoft Edge	80,0 et plus
Apple Safari	14,1+

Navigateur	Version
Samsung Internet	12,0 et plus

Dans la version 3.182.0 ou antérieure, le SDK JavaScript utilise des ES5 artefacts, qui prend en charge les versions minimales suivantes.

Navigateur	Version
Google Chrome	49,0 et plus
Mozilla Firefox	45,0 et plus
Opera	36,0 et plus
Microsoft Edge	12,0 et plus
Windows Internet Explorer	N/A
Apple Safari	9,0 et plus
Navigateur Android	76,0 et plus
Navigateur UC	12,12 et plus
Samsung Internet	5,0 et plus

> **Note**
>
> Des frameworks tels que AWS Amplify celui-ci peuvent ne pas offrir le même support de navigateur que le SDK pour JavaScript. Consultez la AWS Amplify documentation pour plus de détails.

Installez le SDK pour JavaScript

Tous les services ne sont pas immédiatement disponibles dans le SDK ou dans toutes les AWS régions.

Pour installer un service à l' AWS SDK pour JavaScript aide de npm, le gestionnaire de packages Node.js, entrez la commande suivante à l'invite de commande, où *SERVICE* figure le nom du service, tel ques3.

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

Pour obtenir la liste complète des packages de AWS SDK pour JavaScript service client, consultez le guide de référence des AWS SDK pour JavaScript API.

Chargez le SDK pour JavaScript

Après avoir installé le SDK, vous pouvez charger un package client dans votre application de nœud à l'aide `import` de. Par exemple, pour charger le client Amazon S3 et la ListBucketscommande Amazon S3, utilisez ce qui suit.

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

Configurer le SDK pour JavaScript

Avant d'utiliser le SDK pour appeler des services Web JavaScript à l'aide de l'API, vous devez configurer le SDK. Vous devez au minimum configurer :

- La AWS région dans laquelle vous demanderez des services
- Comment votre code s'authentifie auprès de AWS

Outre ces paramètres, vous devrez peut-être également configurer des autorisations pour vos AWS ressources. Par exemple, vous pouvez limiter l'accès à un compartiment Amazon S3 ou restreindre l'accès en lecture seule à une table Amazon DynamoDB.

Le guide de référence AWS SDKs and Tools contient également des paramètres, des fonctionnalités et d'autres concepts fondamentaux communs à de AWS SDKs nombreux.

Les rubriques de cette section décrivent les méthodes de configuration du SDK JavaScript pour Node.js et son JavaScript exécution dans un navigateur Web.

Rubriques

Configuration par service

Vous pouvez configurer le SDK en transmettant les informations de configuration à un objet de service.

La configuration au niveau des services fournit un contrôle significatif sur les services individuels, ce qui vous permet de mettre à jour la configuration des objets de service individuels lorsque vos besoins diffèrent de la configuration par défaut.

> **Note**
>
> Dans la version 2.x du AWS SDK pour JavaScript service, la configuration pouvait être transmise à des constructeurs clients individuels. Cependant, ces configurations seraient d'abord fusionnées automatiquement dans une copie de la configuration `AWS.config` globale du SDK.
> De plus, appeler `AWS.config.update({/* params *})` uniquement la configuration mise à jour pour les clients de service instanciés après l'appel de mise à jour, pas pour les clients existants.
> Ce comportement était une source fréquente de confusion et a rendu difficile l'ajout d'une configuration à l'objet global qui n'affecte qu'un sous-ensemble de clients de service d'une manière compatible avec la transmission. Dans la version 3, il n'existe plus de configuration globale gérée par le SDK. La configuration doit être transmise à chaque client de service instancié. Il est toujours possible de partager la même configuration entre plusieurs clients, mais cette configuration ne sera pas automatiquement fusionnée avec un état global.

Définir la configuration par service

Chaque service que vous utilisez dans le SDK JavaScript est accessible via un objet de service faisant partie de l'API de ce service. Par exemple, pour accéder au service Amazon S3, vous créez l'objet de service Amazon S3. Vous pouvez spécifier les paramètres de configuration spécifiques à un service dans le cadre du constructeur pour cet objet de service.

Par exemple, si vous devez accéder à EC2 des objets Amazon dans plusieurs AWS régions, créez un objet de EC2 service Amazon pour chaque région, puis définissez la configuration régionale de chaque objet de service en conséquence.

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

Définissez la AWS région

Une AWS région est un ensemble nommé de AWS ressources dans la même zone géographique. Un exemple de région est `us-east-1` la région de l'est des États-Unis (Virginie du Nord). Vous spécifiez une région lors de la création d'un client de service dans le SDK pour JavaScript que le SDK accède au service de cette région. Certains services sont disponibles uniquement dans certaines régions.

Le SDK pour JavaScript ne sélectionne pas de région par défaut. Vous pouvez toutefois définir la AWS région à l'aide d'une variable d'environnement ou d'un `config` fichier de configuration partagé.

Dans un constructeur de classe client

Lorsque vous instanciez un objet de service, vous pouvez spécifier la AWS région pour cette ressource dans le cadre du constructeur de classe client, comme illustré ici.

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

Utiliser une variable d'environnement

Vous pouvez définir la région à l'aide de la variable d'environnement `AWS_REGION`. Si vous définissez cette variable, le SDK for la JavaScript lit et l'utilise.

Utiliser un fichier de configuration partagé

Tout comme le fichier d'informations d'identification partagé vous permet de stocker les informations d'identification à utiliser par le SDK, vous pouvez conserver votre AWS région et les autres paramètres de configuration dans un fichier partagé nommé `config` d'après le SDK à utiliser.
Si la variable d'`AWS_SDK_LOAD_CONFIG`environnement est définie sur une valeur vraie, le SDK recherche JavaScript automatiquement un `config` fichier lors de son chargement. L'emplacement d'enregistrement du fichier `config` dépend de votre système d'exploitation :

- Utilisateurs de Linux, macOS ou Unix - `~/.aws/config`
- Utilisateurs de Windows - `C:\Users\USER_NAME\.aws\config`

Si vous n'avez pas encore de fichier `config` partagé, vous pouvez en créer un dans le répertoire désigné. Dans l'exemple suivant, le fichier `config` définit la région et le format de sortie.

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

Pour plus d'informations sur l'utilisation du partage `config` et `credentials` des fichiers, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le Guide de référence des outils AWS SDKs et.

Ordre de priorité pour définir la région

L'ordre de priorité du réglage des régions est le suivant :

1. Si une région est transmise à un constructeur de classe client, cette région est utilisée.
2. Si une région est définie dans la variable d'environnement, cette région est utilisée.
3. Dans le cas contraire, la région définie dans le fichier de configuration partagé est utilisée.

Définir les informations d'identification

AWS utilise des informations d'identification pour identifier qui appelle les services et si l'accès aux ressources demandées est autorisé.

Qu'il soit exécuté dans un navigateur Web ou sur un serveur Node.js, votre JavaScript code doit obtenir des informations d'identification valides avant de pouvoir accéder aux services via l'API. Les informations d'identification peuvent être définies par service, en les transmettant directement à un objet de service.

Il existe plusieurs manières de définir des informations d'identification qui diffèrent entre Node.js et JavaScript dans les navigateurs Web. Les rubriques de cette section décrivent comment définir les informations d'identification dans Node.js ou des navigateurs web. Dans chaque cas, les options sont présentées dans l'ordre recommandé.

Bonnes pratiques en matière d'accréditation

Si les informations d'identification sont correctement définies, le script de votre application ou navigateur peut accéder aux services et ressources nécessaires tout en réduisant l'exposition aux problèmes de sécurité qui peuvent avoir un impact sur les applications stratégiques ou compromettre les données sensibles.

Lors de la définition des informations d'identification, il convient de toujours accorder le moindre privilège requis pour la tâche. Il est plus sûr d'octroyer les autorisations minimales pour vos ressources et d'en ajouter d'autres au besoin, plutôt que d'accorder des autorisations qui dépassent le moindre privilège et, par conséquent, d'avoir à résoudre les problèmes de sécurité que vous pourriez découvrir par la suite. Par exemple, à moins que vous n'ayez besoin de lire et d'écrire des ressources individuelles, telles que des objets dans un compartiment Amazon S3 ou une table DynamoDB, définissez ces autorisations en lecture seule.

Pour plus d'informations sur l'octroi du moindre privilège, consultez la section Accorder le moindre privilège de la rubrique Bonnes pratiques du Guide de l'utilisateur IAM.

Rubriques

Définissez les informations d'identification dans Node.js

Nous recommandons aux nouveaux utilisateurs qui se développent localement et qui n'ont pas reçu de méthode d'authentification de la part de leur employeur de les configurer AWS IAM Identity Center. Pour de plus amples informations, veuillez consulter Authentification du SDK avec AWS.

Dans Node.js, il existe plusieurs façons de fournir vos informations d'identification au kit SDK. Certaines sont plus sécurisées, tandis que d'autres s'avèrent plus pratiques lors du développement d'une application. Lorsque vous obtenez des informations d'identification dans Node.js, veillez à ne pas vous fier à plusieurs sources, telles qu'une variable d'environnement et un fichier JSON que vous chargez. Vous pourriez en effet modifier les autorisations sous lesquelles s'exécute votre code sans vous en rendre compte.

AWS SDK pour JavaScript La version 3 fournit une chaîne de fournisseurs d'informations d'identification par défaut dans Node.js. Vous n'êtes donc pas obligé de fournir un fournisseur d'informations d'identification de manière explicite. La chaîne de fournisseurs d'informations d'identification par défaut tente de résoudre les informations d'identification provenant de différentes sources selon une priorité donnée, jusqu'à ce qu'une information d'identification soit renvoyée par l'une des sources. Vous trouverez la chaîne de fournisseurs d'informations d'identification pour le SDK pour JavaScript V3 ici.

Chaîne de fournisseurs d'identifiants

Tous SDKs ont une série de lieux (ou de sources) qu'ils vérifient afin d'obtenir des informations d'identification valides à utiliser pour faire une demande à un Service AWS. Une fois les informations d’identification valides trouvées, la recherche s’arrête. Cette recherche systématique est appelée chaîne de fournisseurs d'informations d'identification par défaut.

Pour chaque étape de la chaîne, il existe différentes manières de définir les valeurs. La définition des valeurs directement dans le code est toujours prioritaire, suivie de la définition en tant que variables

d'environnement, puis dans le AWS `config` fichier partagé. Pour plus d'informations, consultez la section Priorité des paramètres dans le Guide de référence des outils AWS SDKs et des outils.

Le guide de référence AWS SDKs and Tools contient des informations sur les paramètres de configuration du SDK utilisés par tous AWS SDKs et les AWS CLI. Pour en savoir plus sur la configuration du SDK via le AWS `config` fichier partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés. Pour en savoir plus sur la configuration du SDK en définissant des variables d'environnement, consultez la section Prise en charge des variables d'environnement.

Pour s'authentifier AWS, AWS SDK pour JavaScript vérifie les fournisseurs d'informations d'identification dans l'ordre indiqué dans le tableau suivant.

AWS SDK pour JavaScrip t Méthode du fournisseur d'informations d'identification API Reference par ordre de priorité	Fournisseur (s) d'identification disponible (s)	AWS SDKs Guide de référence et d'outils
`fromEnv()`	AWS clés d'accès à partir de variables d'environnement	AWS clés d'accès
`fromSSO()`	AWS IAM Identity Center. Dans ce guide, consultez Authentification du SDK avec AWS.	Fournisseur d'identifiants IAM Identity Center
`fromIni()`	AWS clés d'accès à partir de fichiers partagés `config` et de `credentials` fichiers	AWS clés d'accès
	Fournisseur d'entités de confiance (tel que`AWS_ROLE_ARN`)	Assumez un rôle IAM
	Jeton d'identité Web provenant de AWS Security Token Service (AWS STS)	Fédérez avec l'identité Web ou OpenID Connect

AWS SDK pour JavaScrip t Méthode du fournisseur d'informations d'identification API Reference par ordre de priorité	Fournisseur (s) d'identification disponible (s)	AWS SDKs Guide de référence et d'outils
	Informations d'identification Amazon Elastic Container Service (Amazon ECS)	Fournisseur d'informations d'identification du conteneur
	Informations d'identification du profil d'instance Amazon Elastic Compute Cloud (Amazon EC2) (fournisseur d'informations d'identification IMDS)	fournisseur d'informations d'identification IMDS
	Fournisseur d'identifiants de processus	Fournisseur d'identifiants de processus
	AWS Centre d'identité IAM	Fournisseur d'identifiants IAM Identity Center
	Fournisseur d'identifiants de connexion	Fournisseur d'identifiants de connexion
`fromLoginCredentials()`	Fournisseur d'identifiants de connexion	Fournisseur d'identifiants de connexion
`fromProcess()`	Fournisseur d'identifiants de processus	Fournisseur d'identifiants de processus
`fromTokenFile()`	Jeton d'identité Web provenant de AWS Security Token Service (AWS STS)	Fédérez avec l'identité Web ou OpenID Connect
`fromContainerMetadata()`	Informations d'identification Amazon Elastic Container Service (Amazon ECS)	Fournisseur d'informations d'identification du conteneur

AWS SDK pour JavaScript Méthode du fournisseur d'informations d'identification API Reference par ordre de priorité	Fournisseur (s) d'identification disponible (s)	AWS SDKs Guide de référence et d'outils
`fromInstanceMetadata()`	Informations d'identification du profil d'instance Amazon Elastic Compute Cloud (Amazon EC2) (fournisseur d'informations d'identification IMDS)	fournisseur d'informations d'identification IMDS

Si vous avez suivi l'approche recommandée pour les nouveaux utilisateurs pour démarrer, vous avez configuré AWS IAM Identity Center l'authentification au cours Authentification du SDK avec AWS de la rubrique Mise en route. D'autres méthodes d'authentification sont utiles dans différentes situations. Pour éviter les risques de sécurité, nous vous recommandons de toujours utiliser des informations d'identification à court terme. Pour les autres procédures relatives aux méthodes d'authentification, voir Authentification et accès dans le guide de référence des outils AWS SDKs et.

Les rubriques de cette section décrivent comment charger les informations d'identification dans Node.js.

Rubriques

- Charger les informations d'identification dans le fichier Node.js à partir des rôles IAM pour Amazon EC2
- Charger les informations d'identification pour une fonction Lambda Node.js

Charger les informations d'identification dans le fichier Node.js à partir des rôles IAM pour Amazon EC2

Si vous exécutez votre application Node.js sur une EC2 instance Amazon, vous pouvez utiliser les rôles IAM pour qu'Amazon EC2 fournisse automatiquement des informations d'identification à l'instance. Si vous configurez votre instance pour utiliser des rôles IAM, le SDK sélectionne automatiquement les informations d'identification IAM pour votre application, éliminant ainsi le besoin de fournir manuellement des informations d'identification.

Pour plus d'informations sur l'ajout de rôles IAM à une EC2 instance Amazon, consultez la section Rôles IAM pour Amazon. EC2

Charger les informations d'identification pour une fonction Lambda Node.js

Lorsque vous créez une AWS Lambda fonction, vous devez créer un rôle IAM spécial autorisé à exécuter la fonction. Il s'agit du rôle d'exécution. Lorsque vous configurez une fonction Lambda, vous devez spécifier le rôle IAM que vous avez créé comme rôle d'exécution correspondant.

Le rôle d'exécution fournit à la fonction Lambda les informations d'identification dont elle a besoin pour s'exécuter et appeler d'autres services Web. Par conséquent, il n'est pas nécessaire de fournir des informations d'identification pour le code Node.js que vous écrivez dans une fonction Lambda.

Pour plus d'informations sur la création d'un rôle d'exécution Lambda, voir Gérer les autorisations : utilisation d'un rôle IAM (rôle d'exécution) dans le Guide du AWS Lambda développeur.

Définir les informations d'identification dans un navigateur Web

Il existe plusieurs façons de fournir vos informations d'identification au kit SDK à partir des scripts de navigateur. Certaines sont plus sécurisées, tandis que d'autres s'avèrent plus pratiques lors du développement d'un script.

Voici comment vous pouvez fournir vos informations d'identification, par ordre de recommandation :

1. Utilisation d'Amazon Cognito Identity pour authentifier les utilisateurs et fournir des informations d'identification
2. Utilisation de l'identité web fédérée

> **Warning**
>
> Nous vous déconseillons de coder en dur vos AWS informations d'identification dans vos scripts. En effet, le codage en dur des informations d'identification risque d'exposer votre ID de clé d'accès et votre clé d'accès secrète.

Rubriques

- Utiliser Amazon Cognito Identity pour authentifier les utilisateurs

Utiliser Amazon Cognito Identity pour authentifier les utilisateurs

La méthode recommandée pour obtenir des AWS informations d'identification pour les scripts de votre navigateur consiste à utiliser le client d'identification Amazon Cognito Identity. `CognitoIdentityClient` Amazon Cognito permet l'authentification des utilisateurs par le biais de fournisseurs d'identité tiers.

Pour utiliser Amazon Cognito Identity, vous devez d'abord créer un pool d'identités dans la console Amazon Cognito. Un groupe d'identités représente le groupe des identités fournies par votre application à vos utilisateurs. Les identités attribuées aux utilisateurs identifient de manière unique chaque compte utilisateur. Les identités Amazon Cognito ne sont pas des informations d'identification. Ils sont échangés contre des informations d'identification à l'aide du support de fédération d'identité Web dans AWS Security Token Service (AWS STS).

Amazon Cognito vous aide à gérer l'abstraction des identités entre plusieurs fournisseurs d'identité. L'identité chargée est ensuite échangée contre les informations d'identification dans AWS STS.

Configuration de l'objet d'identification Amazon Cognito Identity

Si vous n'en avez pas encore créé un, créez un pool d'identités à utiliser avec les scripts de votre navigateur dans la console Amazon Cognito avant de configurer votre client Amazon Cognito. Créez et associez des rôles IAM authentifiés et non authentifiés pour votre pool d'identités. Pour plus d'informations, consultez Tutoriel : Création d'un pool d'identités dans le manuel Amazon Cognito Developer Guide.

L'identité des utilisateurs non authentifiés n'est pas vérifiée, ce rôle est donc approprié pour les utilisateurs invités de votre application ou dans les cas où le fait que l'identité des utilisateurs soit vérifiée n'a pas d'importance. Les utilisateurs authentifiés se connectent à votre application via un fournisseur d'identité tiers qui vérifie leur identité. Assurez-vous de définir de façon appropriée les autorisations des ressources afin de ne pas y accorder l'accès aux utilisateurs non authentifiés.

Après avoir configuré un pool d'identités, utilisez la `fromCognitoIdentityPool` méthode du `@aws-sdk/credential-providers` pour récupérer les informations d'identification du pool d'identités. Dans l'exemple suivant de création d'un client Amazon S3, remplacez-le *`AWS_REGION`* par la région et *`IDENTITY_POOL_ID`* par l'ID du pool d'identités.

```
// Import required AWS SDK clients and command for Node.js
import {S3Client} from "@aws-sdk/client-s3";
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";
```

```
const REGION = AWS_REGION;

const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
 CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: {
            // Optional tokens, used for authenticated login.
        },
  })
});
```

La propriété facultative `logins` est un mappage de noms de fournisseur d'identité avec les jetons d'identité de ces fournisseurs. La façon dont vous obtenez le jeton de la part de votre fournisseur d'identité dépend du fournisseur que vous utilisez. Par exemple, si vous utilisez un groupe d'utilisateurs Amazon Cognito comme fournisseur d'authentification, vous pouvez utiliser une méthode similaire à celle ci-dessous.

```
// 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;
};
```

Passer d'utilisateurs non authentifiés à des utilisateurs authentifiés

Amazon Cognito prend en charge les utilisateurs authentifiés et non authentifiés. Les utilisateurs non authentifiés bénéficient d'un accès à vos ressources, même s'ils ne sont pas connectés avec l'un de vos fournisseurs d'identité. Ce degré d'accès est utile pour afficher du contenu aux utilisateurs avant qu'ils ne se connectent. Chaque utilisateur non authentifié possède une identité unique dans Amazon Cognito, même s'il n'a pas été connecté et authentifié individuellement.

Utilisateur initialement non authentifié

Les utilisateurs commencent généralement par le rôle non authentifié, pour lequel vous définissez la propriété des informations d'identification de votre objet de configuration sans propriété `logins`. Dans ce cas, vos informations d'identification par défaut peuvent ressembler à ce qui suit :

```
// Import the required AWS SDK pour 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.
});
```

Basculement vers un utilisateur authentifié

Lorsqu'un utilisateur non authentifié se connecte à un fournisseur d'identité et que vous disposez d'un jeton, vous pouvez passer du statut d'utilisateur non authentifié à celui d'utilisateur authentifié en appelant une fonction personnalisée qui met à jour l'objet d'identification et ajoute le jeton. `logins`

```
// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
  creds.params.Logins = creds.params.logins || {};
  creds.params.Logins[providerName] = token;

  // Expire credentials to refresh them on the next request
  creds.expired = true;
}
```

Considérations relatives à Node.js

Bien que le code Node.js le soit JavaScript, l'utilisation du fichier AWS SDK pour JavaScript dans Node.js peut être différente de l'utilisation du SDK dans les scripts de navigateur. Certaines méthodes d'API fonctionnent dans Node.js, mais pas dans les scripts du navigateur et inversement. Et l'utilisation réussie de certains d'entre eux APIs dépend de votre connaissance des modèles de codage courants de Node.js, tels que l'importation et l'utilisation d'autres modules Node.js tels que le `File System (fs)` module.

> **Note**
>
> AWS recommande d'utiliser la version Active LTS de Node.js pour le développement.

Utiliser les modules Node.js intégrés

Node.js fournit un ensemble de modules intégrés que vous pouvez utiliser sans les installer. Pour utiliser ces modules, créez un objet en appliquant la méthode `require` afin de nommer le module. Par exemple, pour inclure le module HTTP intégré, utilisez le code ci-dessous.

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

Invoquez les méthodes du module comme si elles étaient des méthodes de cet objet. Par exemple, voici un code qui lit un fichier HTML.

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

Pour une liste complète de tous les modules intégrés fournis par Node.js, consultez la documentation Node.js sur le site Web Node.js.

Utiliser les packages npm

Outre les modules intégrés, vous pouvez également inclure et intégrer du code tiers à partir du npm gestionnaire de packages Node.js. Il s'agit d'un référentiel de packages Node.js open source et d'une interface de ligne de commande permettant d'installer ces packages. Pour plus d'informations npm et une liste des packages actuellement disponibles, consultez https://www.npmjs.com. Vous pouvez également en savoir plus sur les packages Node.js supplémentaires que vous pouvez utiliser ici GitHub.

Configurer MaxSockets dans Node.js

Dans Node.js, vous pouvez définir le nombre maximal de connexions par origine. Si `maxSockets` est défini, le client HTTP de bas niveau place les demandes en file d'attente et les affecte aux sockets au fur et à mesure qu'ils deviennent disponibles.

Vous pouvez ainsi définir un nombre maximal supérieur de demandes simultanées pour une origine donnée. Le fait de réduire cette valeur peut réduire le nombre d'erreurs de limitation ou d'expiration reçues. Toutefois, il peut aussi en résulter une utilisation accrue de la mémoire, car les demandes sont placées en file d'attente jusqu'à ce qu'un socket devienne disponible.

L'exemple suivant montre comment effectuer une configuration `maxSockets` pour un client DynamoDB.

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

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

Le SDK pour JavaScript utilise une `maxSockets` valeur de 50 si vous ne fournissez pas de valeur ou d'`Agent`objet. Si vous fournissez un `Agent` objet, sa `maxSockets` valeur sera utilisée. Pour plus d'informations sur `maxSockets` le paramétrage dans Node.js, consultez la documentation Node.js.

À partir de la version 3.521.0 du AWS SDK pour JavaScript, vous pouvez utiliser la syntaxe abrégée suivante pour configurer. `requestHandler`

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

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

Réutiliser les connexions avec keep-alive dans Node.js

L'agent HTTP/HTTPS Node.js par défaut crée une nouvelle connexion TCP pour chaque nouvelle demande. Pour éviter les coûts liés à l'établissement d'une nouvelle connexion, les connexions AWS SDK pour JavaScript TCP sont réutilisées par défaut.

Pour les opérations de courte durée, telles que les requêtes Amazon DynamoDB, le temps de latence associé à la configuration d'une connexion TCP peut être supérieur à celui de l'opération elle-même. En outre, étant donné que le chiffrement DynamoDB au repos est intégré, vous pouvez AWS KMSrencontrer des latences lorsque la base de données doit rétablir de AWS KMS nouvelles entrées de cache pour chaque opération.

Si vous ne souhaitez pas réutiliser les connexions TCP, vous pouvez désactiver la réutilisation active de ces connexions, client par service, comme indiqué dans l'exemple suivant pour un client DynamoDB. `keepAlive`

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

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

Si cette option `keepAlive` est activée, vous pouvez également définir le délai initial pour les paquets TCP Keep-Alive`keepAliveMsecs`, qui est par défaut de 1 000 ms. Consultez la documentation Node.js pour plus de détails.

Configurer les proxys pour Node.js

Si vous ne pouvez pas vous connecter directement à Internet, le SDK JavaScript prend en charge l'utilisation de proxys HTTP ou HTTPS via un agent HTTP tiers.

Pour trouver un agent HTTP tiers, recherchez « proxy HTTP » sur npm.

Pour installer un agent proxy HTTP tiers, entrez le code suivant à l'invite de commande, où *PROXY* figure le nom du `npm` package.

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

Pour utiliser un proxy dans votre application, utilisez la `httpsAgent` propriété `httpAgent` and, comme illustré dans l'exemple suivant pour un client DynamoDB.

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { HttpsProxyAgent } from "hpagent";
const agent = new HttpsProxyAgent({ proxy: "http://internal.proxy.com" });
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: agent,
        httpsAgent: agent
    }),
});
```

> **Note**
>
> `httpAgent`n'est pas la même chose que`httpsAgent`, et comme la plupart des appels du client seront adressés à`https`, les deux doivent être définis.

Enregistrez les ensembles de certificats dans Node.js

Les magasins de confiance par défaut pour Node.js incluent les certificats nécessaires pour accéder aux AWS services. Dans certains cas, il peut être préférable d'inclure uniquement un ensemble de certificats donné.

Dans cet exemple, un certificat spécifique sur le disque est utilisé pour créer un `https.Agent` qui rejette les connexions, à moins que le certificat désigné ne soit fourni. La nouvelle création `https.Agent` est ensuite utilisée par le client DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";
import { readFileSync } from "fs";
const certs = [readFileSync("/path/to/cert.pem")];
const agent = new Agent({
  rejectUnauthorized: true,
  ca: certs
});
const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  })
});
```

Considérations à propos des scripts du navigateur

Les rubriques suivantes décrivent les considérations particulières relatives à l'utilisation des scripts AWS SDK pour JavaScript dans le navigateur.

Rubriques

- Créez le SDK pour les navigateurs
- Partage des ressources cross-origin (CORS)
- Regroupez des applications avec Webpack

Créez le SDK pour les navigateurs

Contrairement au SDK pour JavaScript la version 2 (V2), la version 3 n'est pas fournie sous forme de JavaScript fichier avec prise en charge d'un ensemble de services par défaut. Au lieu de cela, la V3 vous permet de regrouper et d'inclure dans le navigateur uniquement le SDK pour les JavaScript fichiers dont vous avez besoin, ce qui réduit les frais de gestion. Nous vous recommandons d'utiliser Webpack pour regrouper le SDK requis pour JavaScript les fichiers, ainsi que tous les packages tiers supplémentaires dont vous avez besoin, dans un seul `Javascript` fichier, et le charger dans

des scripts de navigateur à l'aide d'une `<script>` balise. Pour plus d'informations sur Webpack, consultezRegroupez des applications avec Webpack.

Si vous travaillez avec le SDK en dehors d'un environnement qui applique CORS dans votre navigateur et si vous souhaitez accéder à tous les services fournis par le SDK pour JavaScript, vous pouvez créer une copie personnalisée du SDK localement en clonant le référentiel et en exécutant les mêmes outils de génération que ceux utilisés pour créer la version hébergée par défaut du SDK. Les sections suivantes décrivent les étapes à suivre pour créer le kit SDK avec des services et des versions d'API supplémentaires.

Utilisez le SDK Builder pour créer le SDK pour JavaScript

> **Note**
>
> La version 3 (V3) d'Amazon Web Services ne prend plus en charge Browser Builder. Pour minimiser l'utilisation de la bande passante par les applications du navigateur, nous vous recommandons d'importer des modules nommés et de les regrouper afin de réduire leur taille. Pour plus d'informations sur le regroupement, consultezRegroupez des applications avec Webpack.

Partage des ressources cross-origin (CORS)

Le partage des ressources cross-origin, ou CORS, est une fonctionnalité de sécurité des navigateurs web modernes. Elle permet aux navigateurs web de négocier les domaines pouvant effectuer des demandes de sites web ou services externes.

CORS est une fonction importante à prendre en considération lors du développement des applications de navigateur avec le kit AWS SDK pour JavaScript , car la plupart des demandes de ressources sont envoyées à un domaine externe, tel que le point de terminaison d'un service web. Si votre JavaScript environnement applique la sécurité CORS, vous devez configurer CORS avec le service.

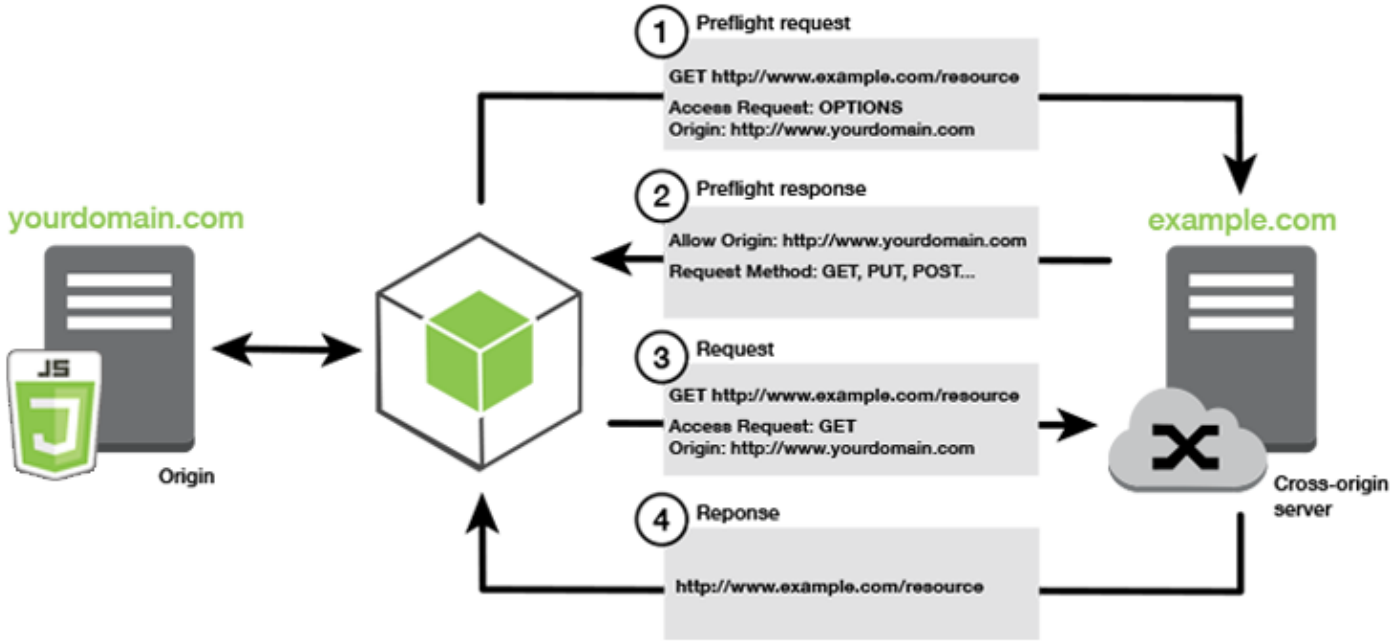
CORS détermine s'il convient d'autoriser le partage de ressources dans une demande d'origine croisée sur la base des éléments suivants :

- Le domaine spécifique qui effectue la demande
- Le type de demande HTTP effectuée (GET, PUT, POST, DELETE, etc.)

Comment fonctionne CORS

Dans le cas le plus simple, votre script de navigateur effectue une demande GET pour une ressource à partir d'un serveur appartenant à un autre domaine. Selon la configuration CORS de ce serveur, si la demande provient d'un domaine qui est autorisé à soumettre des demandes GET, le serveur cross-origin répond en retournant la ressource demandée.

Si le domaine effectuant la demande ou si le type de demande HTTP n'est pas autorisé, la demande est refusée. Toutefois, CORS permet de vérifier la demande en amont avant de la soumettre. Dans ce cas, une demande en amont est effectuée. Cette demande inclut l'envoi de l'opération de demande d'accès `OPTIONS`. Si la configuration de la fonction CORS du serveur cross-origin accorde l'accès au domaine demandeur, le serveur renvoie une réponse en amont qui répertorie tous les types de requête HTTP que le domaine demandeur peut faire sur la ressource demandée.



La configuration CORS est-elle requise ?

Les compartiments Amazon S3 nécessitent une configuration CORS avant que vous puissiez effectuer des opérations sur ceux-ci. Dans certains JavaScript environnements, le CORS peut ne pas être appliqué et sa configuration n'est donc pas nécessaire. Par exemple, si vous hébergez votre application à partir d'un compartiment Amazon S3 et que vous accédez à des ressources depuis un point de terminaison spécifique `*.s3.amazonaws.com` ou depuis un autre point de terminaison spécifique, vos demandes n'accèderont pas à un domaine externe. Par conséquent,

cette configuration n'exige pas CORS. Dans ce cas, CORS est toujours utilisé pour des services autres qu'Amazon S3.

Configurer CORS pour un compartiment Amazon S3

Vous pouvez configurer un compartiment Amazon S3 pour utiliser CORS dans la console Amazon S3.

Si vous configurez CORS dans la console de gestion des services AWS Web, vous devez utiliser JSON pour créer une configuration CORS. La nouvelle console de gestion des services AWS Web prend uniquement en charge les configurations JSON CORS.

> **Important**
>
> Dans la nouvelle console de gestion des services AWS Web, la configuration CORS doit être JSON.

1. Dans la console de gestion des services AWS Web, ouvrez la console Amazon S3, recherchez le compartiment que vous souhaitez configurer et cochez sa case.
2. Dans le volet qui s'ouvre, choisissez Permissions.
3. Dans l'onglet Autorisation, choisissez Configuration CORS.
4. Entrez votre configuration CORS dans l'éditeur de configuration CORS, puis choisissez Enregistrer.

Une configuration CORS est un fichier XML qui contient une série de règles au sein d'un élément `<CORSRule>`. Une configuration peut contenir jusqu'à 100 règles. Une règle est définie par l'une des balises suivantes :

- `<AllowedOrigin>`— Spécifie les origines de domaines que vous autorisez à effectuer des demandes entre domaines.
- `<AllowedMethod>`— Spécifie le type de demande que vous autorisez (GET, PUT, POST, DELETE, HEAD) dans les requêtes interdomaines.
- `<AllowedHeader>`— Spécifie les en-têtes autorisés dans une demande de prévol.

Pour des exemples de configurations, voir Comment configurer CORS sur mon bucket ? dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Exemple de configuration CORS

L'exemple de configuration CORS suivant permet à un utilisateur de visualiser, d'ajouter, de supprimer ou de mettre à jour des objets à l'intérieur d'un compartiment du domaine`example.org`. Cependant, nous vous recommandons de `<AllowedOrigin>` définir le domaine de votre site Web. Vous pouvez spécifier `"*"` pour autoriser n'importe quelle origine.

> **Important**
>
> Dans la nouvelle console S3, la configuration CORS doit être de type JSON.

XML

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

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
```

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

Cette configuration n'autorise pas l'utilisateur à effectuer des actions sur le compartiment. Il active le modèle de sécurité du navigateur pour autoriser une demande à Amazon S3. Les autorisations doivent être configurées via des autorisations de compartiment ou des autorisations de rôle IAM.

Vous pouvez l'utiliser `ExposeHeader` pour permettre au SDK de lire les en-têtes de réponse renvoyés par Amazon S3. Par exemple, si vous lisez l'`ETag`en-tête d'un téléchargement partitionné PUT ou partiel, vous devez inclure la `ExposeHeader` balise dans votre configuration, comme indiqué dans l'exemple précédent. Le kit SDK ne peut accéder qu'aux en-têtes qui sont exposés via la configuration CORS. Si vous définissez des métadonnées sur l'objet, les valeurs sont renvoyées sous forme d'en-têtes avec le préfixe `x-amz-meta-`, comme `x-amz-meta-my-custom-header` par exemple, et doivent également être exposées de la même manière.

Regroupez des applications avec Webpack

L'utilisation de modules de code par des applications Web dans des scripts de navigateur ou dans Node.js crée des dépendances. Ces modules de code peuvent avoir leurs propres dépendances, ce qui se traduit par un ensemble de modules interconnectés nécessaires à votre application pour fonctionner. Pour gérer les dépendances, vous pouvez utiliser un bundler de modules tel que`webpack`.

Le `webpack` module bundler analyse le code de votre application, à la recherche d'`require`instructions `import` ou d'instructions, pour créer des ensembles contenant tous les actifs dont votre application a besoin. Cela permet de diffuser facilement les actifs via une page Web. Le SDK pour JavaScript peut être inclus `webpack` comme l'une des dépendances à inclure dans le bundle de sortie.

Pour plus d'informations sur`webpack`, consultez le bundler du module Webpack sur. GitHub

Installer webpack

Pour installer le bundler de `webpack` modules, vous devez d'abord installer npm, le gestionnaire de packages Node.js. Tapez la commande suivante pour installer la `webpack` CLI et le JavaScript module.

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

Pour utiliser le `path` module permettant de travailler avec les chemins de fichiers et de répertoires, qui est installé automatiquement avec Webpack, vous devrez peut-être installer le `path-browserify` package Node.js.

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

Configurer le webpack

Par défaut, Webpack recherche un JavaScript fichier nommé `webpack.config.js` dans le répertoire racine de votre projet. Ce fichier spécifie vos options de configuration. Voici un exemple de fichier de `webpack.config.js` configuration pour les WebPack versions 5.0.0 et ultérieures.

> **Note**
>
> Les exigences de configuration du Webpack varient en fonction de la version du Webpack que vous installez. Pour plus d'informations, consultez la documentation du Webpack.

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

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

Dans cet exemple, `browser.js` est spécifié comme point d'entrée. Le point d'entrée est le fichier `webpack` utilisé pour commencer à rechercher les modules importés. Le nom de fichier de la sortie est spécifié en tant que `bundle.js`. Ce fichier de sortie contiendra tout ce dont JavaScript l'application a besoin pour fonctionner. Si le code spécifié dans le point d'entrée importe ou nécessite d'autres modules, tels que le SDK pour JavaScript, ce code est groupé sans qu'il soit nécessaire de le spécifier dans la configuration.

Exécuter le webpack

Pour créer une application à utiliser`webpack`, ajoutez ce qui suit à l'`scripts`objet de votre `package.json` fichier.

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

Voici un exemple de `package.json` fichier illustrant l'ajout`webpack`.

```
{
  "name": "aws-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@aws-sdk/client-iam": "^3.32.0",
```

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

Pour créer votre application, entrez la commande suivante.

```
npm run build
```

Le `webpack` module bundler génère ensuite le JavaScript fichier que vous avez spécifié dans le répertoire racine de votre projet.

Utiliser le bundle Webpack

Pour utiliser le bundle dans un script de navigateur, vous pouvez l'intégrer à l'aide d'une `<script>` balise, comme illustré dans l'exemple suivant.

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

Bundle pour Node.js

Vous pouvez l'utiliser `webpack` pour générer des bundles qui s'exécutent dans Node.js en les spécifiant `node` comme cible dans la configuration.

```
target: "node"
```

Cela s'avère utile lors de l'exécution d'une application Node.js dans un environnement où l'espace disque est limité. Voici un exemple de configuration `webpack.config.js` avec Node.js spécifié comme cible de sortie.

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

Utilisez les AWS services du SDK pour JavaScript

La AWS SDK pour JavaScript v3 donne accès aux services qu'elle prend en charge par le biais d'un ensemble de classes de clients. À partir de ces classes client, vous créez des objets d'interface de service, couramment appelés objets de service. Chaque AWS service pris en charge possède une ou plusieurs classes de clients qui offrent un faible niveau d'utilisation APIs des fonctionnalités et des ressources du service. Par exemple, Amazon APIs DynamoDB est disponible via la classe. `DynamoDB`

Les services exposés via le SDK JavaScript suivent le modèle demande-réponse pour échanger des messages avec les applications d'appel. Dans ce modèle, le code qui appelle un service envoie une demande HTTP/HTTPS à un point de terminaison du service. La demande contient les paramètres nécessaires pour appeler avec succès la fonction spécifique appelée. Le service qui est appelé génère une réponse qui est renvoyée au demandeur. La réponse contient des données si l'opération a abouti ou les informations relatives à l'erreur si l'opération n'a pas abouti.

L'appel d'un AWS service inclut le cycle de vie complet des demandes et des réponses d'une opération sur un objet de service, y compris les tentatives de nouvelle tentative. Une requête contient zéro ou plusieurs propriétés sous forme de paramètres JSON. La réponse est encapsulée dans un objet lié à l'opération et est renvoyée au demandeur via l'une des nombreuses techniques, telles qu'une fonction de rappel ou une promesse. JavaScript

Rubriques

Création et appel d'objets de service

L' JavaScript API prend en charge la plupart AWS des services disponibles. Chaque service de l' JavaScriptAPI fournit à une classe client une `send` méthode que vous pouvez utiliser pour appeler toutes les API prises en charge par le service. Pour plus d'informations sur les classes de service, les opérations et les paramètres de l' JavaScript API, consultez la référence de l'API.

Lorsque vous utilisez le SDK dans Node.js, vous ajoutez le package SDK pour chaque service dont vous avez besoin à l'application que vous utilisez`import`, qui prend en charge tous les services actuels. L'exemple suivant crée un objet de service Amazon S3 dans la `us-west-1` région.

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

Spécifier les paramètres de l'objet de service

Lorsque vous appelez une méthode d'un objet de service, transmettez des paramètres au format JSON, comme requis par l'API. Par exemple, dans Amazon S3, pour obtenir un objet pour un compartiment et une clé spécifiques, transmettez les paramètres suivants à la `GetObjectCommand` méthode à partir du`S3Client`. Pour plus d'informations sur la transmission de paramètres JSON, consultez Travailler avec JSON.

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

Pour plus d'informations sur les paramètres Amazon S3, consultez @aws -sdk/client-s3 dans le guide de référence des API.

Utilisez @smithy /types pour les clients générés dans TypeScript

Si vous l'utilisez TypeScript, le `@smithy/types` package vous permet de manipuler les formes d'entrée et de sortie d'un client.

Scénario : supprimer **`undefined`** des structures d'entrée et de sortie

Les membres des formes générées sont unifiés `undefined` pour les formes d'entrée et sont ? (facultatifs) pour les formes de sortie. Pour les entrées, cela reporte la validation au service. Pour

les sorties, cela suggère fortement que vous devriez vérifier les données de sortie au moment de l'exécution.

Si vous souhaitez ignorer ces étapes, utilisez les aides `AssertiveClient` ou `UncheckedClient` tapez. L'exemple suivant utilise les assistants de type avec le service Amazon S3.

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

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

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

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

Lorsque vous utilisez la transformation sur un client non agrégé avec la `Command` syntaxe, l'entrée ne peut pas être validée car elle passe par une autre classe, comme indiqué dans l'exemple ci-dessous.

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

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

const list = await s3.send(
  new ListBucketsCommand({
    // command inputs are not validated by the type transform.
```

```
    // because this is a separate class.
  })
);

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

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

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

Scénario : réduction des types de blob de charge utile de sortie d'un client TypeScript généré par Smithy

Ce scénario est principalement pertinent pour les opérations avec des organismes de streaming, comme `S3Client` dans la AWS SDK pour JavaScript v3.

Étant donné que les types de charge utile blob dépendent de la plate-forme, vous souhaiterez peut-être indiquer dans votre application qu'un client s'exécute dans un environnement spécifique. Cela permet de réduire les types de charge utile blob, comme indiqué dans l'exemple suivant.

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

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

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

```
// The default type of blob payloads is a wide union type including multiple possible
// request handlers.
const body1: StreamingBlobPayloadOutputTypes = (await s3Default.send(new
 GetObjectCommand({ Key: "", Bucket: "" })))
  .Body!;

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

Appeler les services de manière asynchrone

Toutes les demandes effectuées via le kit SDK sont asynchrones. Il est important de garder cela à l'esprit lorsque vous écrivez des scripts de navigateur. JavaScript l'exécution dans un navigateur Web ne comporte généralement qu'un seul thread d'exécution. Après avoir effectué un appel asynchrone à un AWS service, le script du navigateur continue de s'exécuter et peut ainsi essayer d'exécuter du code qui dépend de ce résultat asynchrone avant qu'il ne soit renvoyé.

Les appels asynchrones à un AWS service incluent la gestion de ces appels afin que votre code n'essaie pas d'utiliser des données avant qu'elles ne soient disponibles. Les rubriques de cette section expliquent pourquoi il est nécessaire de gérer les appels asynchrones et détaillent les différentes techniques de gestion disponibles.

Bien que vous puissiez utiliser n'importe laquelle de ces techniques pour gérer les appels asynchrones, nous vous recommandons d'utiliser async/await pour tout nouveau code.

async/wait

Nous vous recommandons d'utiliser cette technique car il s'agit du comportement par défaut dans la version 3.

promettre

Utilisez cette technique dans les navigateurs qui ne prennent pas en charge le mode async/await.

rappel

Évitez d'utiliser des rappels, sauf dans des cas très simples. Toutefois, cela peut vous être utile pour les scénarios de migration.

Rubriques

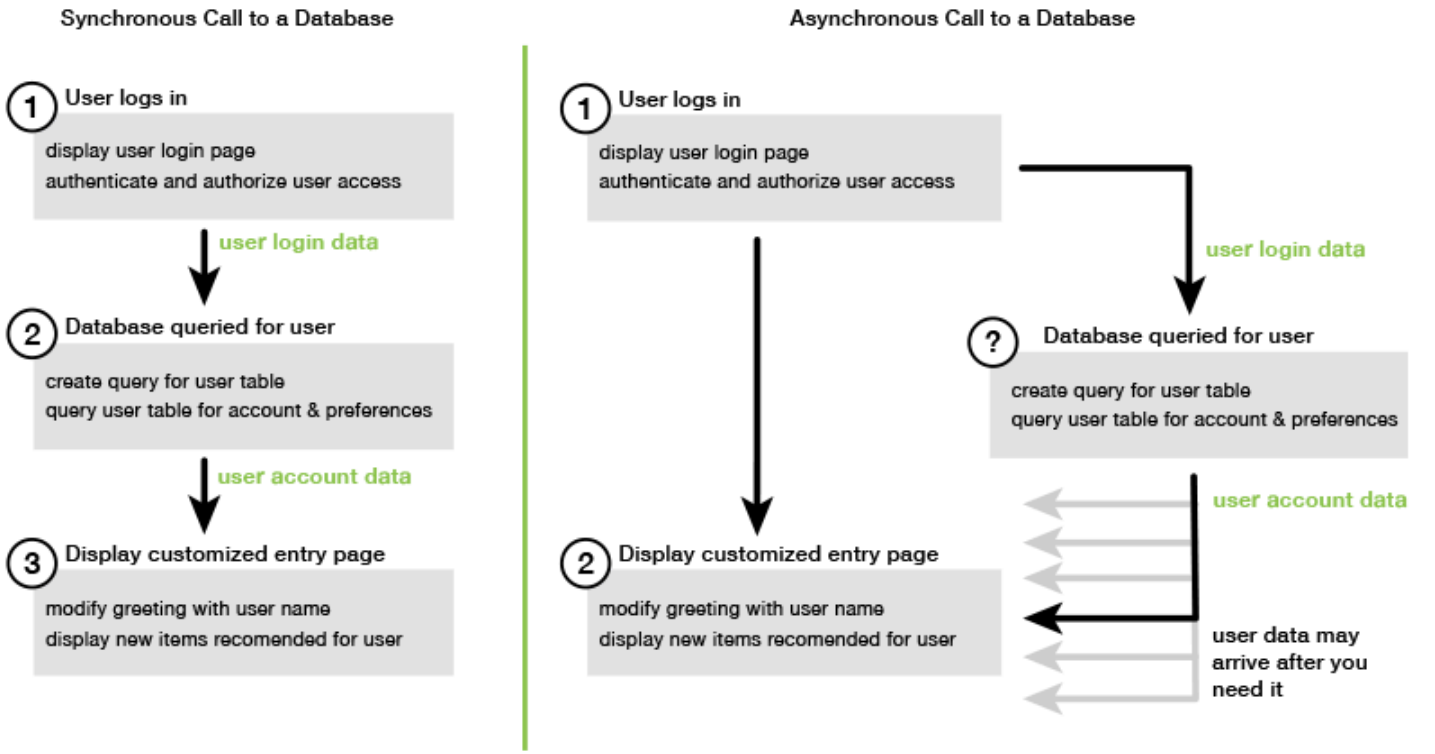
Gérer les appels asynchrones

Par exemple, la page d'accueil d'un site e-commerce autorise le renvoi de la connexion des utilisateurs. Pour les clients qui se connectent, cela présente notamment l'avantage, après connexion, de disposer d'une personnalisation en fonction de leurs préférences. Pour cela :

1. Le client doit se connecter et être validé avec ses identifiants de connexion.
2. La demande des préférences du client est faite auprès d'une base de données client.
3. La base de données fournit les préférences du client qui sont utilisées pour personnaliser le site avant le chargement de la page.

Si ces tâches s'exécutent de façon synchrone, chaque tâche doit se terminer avant le démarrage de la suivante. Le chargement de la page Web ne pourra pas être terminé tant que les préférences du client ne seront pas renvoyées de la base de données. Cependant, une fois que la requête de base de données est envoyée au serveur, la réception des données client peut être retardée ou même échouer en raison d'un goulot d'étranglement, d'un trafic de base de données exceptionnellement dense ou d'une mauvaise connexion d'un appareil mobile.

Pour éviter que le site Web ne se bloque dans ces conditions, appelez la base de données de manière asynchrone. Après l'exécution de l'appel de base de données, si vous envoyez une demande asynchrone, votre code continue de s'exécuter comme prévu. Si vous ne gérez pas correctement la réponse d'un appel asynchrone, votre code peut tenter d'utiliser les informations attendues de la base de données alors que ces données ne sont pas encore disponibles.



Utiliser async/await

Plutôt que d'utiliser des promesses, vous devriez envisager d'utiliser async/await. Les fonctions asynchrones sont plus simples à utiliser que les promesses. Await ne peut être utilisé que dans une fonction asynchrone pour attendre une valeur de manière asynchrone.

L'exemple suivant utilise async/await pour répertorier toutes vos tables Amazon DynamoDB dans. `us-west-2`

> **Note**
>
> Pour exécuter cet exemple :
>
> - Installez le client AWS SDK pour JavaScript DynamoDB en `npm install @aws-sdk/client-dynamodb` entrant dans la ligne de commande de votre projet.
> - Assurez-vous d'avoir correctement configuré vos AWS informations d'identification. Pour de plus amples informations, veuillez consulter Définir les informations d'identification.

```
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**
>
> Tous les navigateurs ne prennent pas en charge le mode async/await. Consultez la section Fonctions asynchrones pour une liste des navigateurs compatibles avec async/await.

JavaScript Promesses d'utilisation

Utilisez la méthode AWS SDK pour JavaScript v3 (`ListTablesCommand`) du client de service pour effectuer l'appel de service et gérer le flux asynchrone au lieu d'utiliser des rappels. L'exemple suivant montre comment obtenir les noms de vos tables Amazon DynamoDB. `us-west-2`

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbClient = new DynamoDBClient({ region: 'us-west-2' });

dbClient.listtables(new ListTablesCommand({}))
  .then(response => {
    console.log(response.TableNames.join('\n'));
  })
  .catch((error) => {
    console.error(error);
  });
```

Coordonner plusieurs promesses

Dans certains cas, votre code doit effectuer plusieurs appels asynchrones qui nécessitent une action uniquement lorsqu'ils sont tous renvoyés avec succès. Si vous gérez ces appels de méthode asynchrone individuels avec des promesses, vous pouvez créer une autre promesse qui utilise la méthode `all`.

Cette méthode exécute cette promesse générique si et quand la série de promesses que vous transmettez dans la méthode est exécutée. La fonction de rappel reçoit une série des valeurs des promesses transmises à la méthode `all`.

Dans l'exemple suivant, une AWS Lambda fonction doit effectuer trois appels asynchrones à Amazon DynamoDB, mais elle ne peut se terminer que lorsque les promesses pour chaque appel ont été remplies.

```
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;
```

Support du navigateur et de Node.js pour les promesses

Support pour les JavaScript promesses natives (ECMAScript 2015) dépend du JavaScript moteur et de la version dans lesquels votre code s'exécute. Pour déterminer la prise en charge des JavaScript promesses dans chaque environnement dans lequel votre code doit être exécuté, consultez le tableau de ECMAScript compatibilité sur GitHub.

Utiliser une fonction de rappel anonyme

Chaque méthode d'objet de service peut accepter une fonction de rappel anonyme comme dernier paramètre. La signature de cette fonction de rappel est la suivante.

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

Cette fonction de rappel est exécutée lorsqu'une réponse positive ou des données d'erreur sont renvoyées. Si l'appel de méthode aboutit, le contenu de la réponse est disponible pour la fonction de rappel dans le paramètre `data`. Si l'appel n'aboutit pas, les détails relatifs à l'échec sont disponibles dans le paramètre `error`.

En général, le code à l'intérieur de la fonction de rappel effectue un test afin d'identifier une éventuelle erreur. Si une erreur est renvoyée, elle est traitée par le code. Si aucune erreur n'est renvoyée, le code récupère les données dans la réponse du paramètre `data`. La forme de base de la fonction de rappel ressemble à cet exemple.

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

Dans l'exemple précédent, les détails de l'erreur ou ceux des données renvoyées sont consignés dans la console. Voici un exemple illustrant une fonction de rappel transmise dans le cadre de l'appel d'une méthode sur un objet de service.

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

Création de demandes de service pour les clients

Il est simple de faire des demandes aux clients de AWS service. La version 3 (V3) du SDK pour vous JavaScript permet d'envoyer des demandes.

> **Note**
>
> Vous pouvez également effectuer des opérations à l'aide des commandes de la version 2 (V2) lorsque vous utilisez la version 3 du SDK pour. JavaScript Pour de plus amples informations, veuillez consulter Utilisation des commandes v2.

Pour envoyer une demande :

1. Initialisez un objet client avec la configuration souhaitée, telle qu'une AWS région spécifique.
2. (Facultatif) Créez un objet JSON de demande avec les valeurs de la demande, telles que le nom d'un compartiment Amazon S3 spécifique. Vous pouvez examiner les paramètres de la demande en consultant la rubrique de référence de l'API relative à l'interface dont le nom est associé à la méthode client. Par exemple, si vous utilisez la méthode *`AbcCommand`* client, l'interface de demande est*`AbcInput`*.
3. Initialisez une commande de service, éventuellement, avec l'objet de demande en entrée.
4. Appelez `send` le client avec l'objet de commande en entrée.

Par exemple, pour répertorier vos tables Amazon DynamoDB, vous pouvez `us-west-2` le faire avec async/await.

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

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

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

Gérer les réponses des clients du service

Une fois qu'une méthode client de service a été appelée, elle renvoie une instance d'objet de réponse d'une interface dont le nom est associé à la méthode client. Par exemple, si vous utilisez la méthode *`AbcCommand`* client, l'objet de réponse est de type *`AbcResponse`* (interface).

Données d'accès renvoyées dans la réponse

L'objet de réponse contient les données, sous forme de propriétés, renvoyées par la demande de service.

DansCréation de demandes de service pour les clients, la `ListTablesCommand` commande a renvoyé les noms des tables dans la `TableNames` propriété de la réponse.

Informations sur les erreurs d'accès

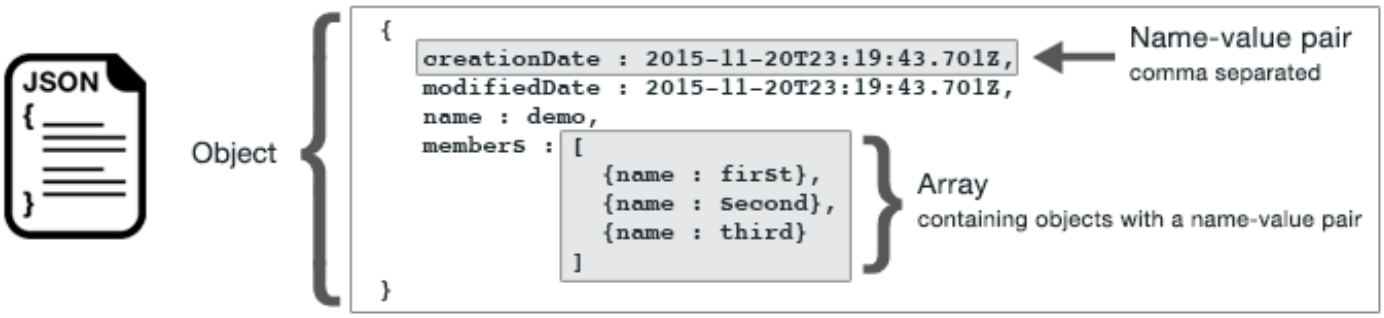
Si une commande échoue, elle lance une exception. L'extrait de code suivant montre comment gérer une exception de service.

```
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
  }
}
```

Travailler avec JSON

Le format JSON est un format d'échange de données à la fois lisible par l'homme et lisible par machine. Bien que le nom JSON soit l'acronyme de JavaScript Object Notation, le format JSON est indépendant de tout langage de programmation.

Il AWS SDK pour JavaScript utilise le JSON pour envoyer des données aux objets de service lorsqu'il fait des demandes et reçoit les données des objets de service au format JSON. Pour plus d'informations sur le format JSON, consultez json.org.



JSON représente les données de deux manières :

- En tant qu'objet, qui est une collection non ordonnée de paires nom-valeur. Un objet est défini entre des accolades gauche ({) et droite (}). Chaque paire nom-valeur commence par le nom, suivi de deux points, suivi de la valeur. Les paires nom-valeur sont séparées par des virgules.
- En tant que tableau, qui est une collection ordonnée de valeurs. Une série est définie entre crochets gauche ([) et droit (]). Les éléments de la série sont séparés par des virgules.

Voici un exemple d'un objet JSON contenant une série d'objets dans laquelle les objets représentent des cartes d'un jeu. Chaque carte est définie par deux paires nom-valeur. L'une spécifie une valeur unique qui identifie cette carte et l'une autre spécifie une URL qui pointe vers l'image de la carte correspondante.

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

JSON en tant que paramètres d'objet de service

Voici un exemple de JSON simple utilisé pour définir les paramètres d'un appel à un objet AWS Lambda de service.

```
const params = {
```

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

L'objet `params` est défini par trois paires nom-valeur, séparées par des virgules et entourées d'accolades gauche et droite. Lorsque vous fournissez des paramètres à un appel de méthode d'objet de service, les noms sont déterminés par les noms de paramètres pour la méthode d'objet de service que vous prévoyez d'appeler. Lors de l'appel d'une fonction Lambda`FunctionName`,,`Payload`, `LogType` et sont les paramètres utilisés pour appeler la méthode sur un `invoke` objet de service Lambda.

Lorsque vous transmettez des paramètres à un appel de méthode d'objet de service, fournissez l'objet JSON à l'appel de méthode, comme illustré dans l'exemple suivant d'appel d'une fonction Lambda.

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

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

Enregistrement AWS SDK pour JavaScript des appels

AWS SDK pour JavaScript Il est équipé d'un enregistreur intégré qui vous permet de consigner les appels d'API que vous effectuez avec le SDK pour. JavaScript

Pour activer l'enregistreur et imprimer les entrées du journal dans la console, configurez le client de service à l'aide du `logger` paramètre facultatif. L'exemple ci-dessous active la journalisation du client tout en ignorant les résultats de trace et de débogage.

```
new S3Client({
```

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

Utilisation d'un intergiciel pour enregistrer les demandes

Il AWS SDK pour JavaScript utilise une pile intergicielle pour contrôler le cycle de vie d'un appel d'opération. Chaque intergiciel de la pile appelle le middleware suivant après avoir apporté des modifications à l'objet de la requête. Cela facilite également le débogage des problèmes dans la pile, car vous pouvez voir exactement quel intergiciel a été appelé à l'origine d'une erreur. Voici un exemple de journalisation des demandes à l'aide d'un intergiciel :

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

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

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

Dans l'exemple ci-dessus, un intergiciel est ajouté à la pile d'intergiciels du client DynamoDB. Le premier argument est une fonction qui accepte`next`, le prochain intergiciel de la pile à appeler et `context` un objet contenant des informations sur l'opération appelée. Il renvoie une fonction qui accepte`args`, un objet contenant les paramètres transmis à l'opération et à la demande, et il renvoie le résultat de l'appel du prochain intergiciel avec. `args`

Utiliser des points de terminaison AWS basés sur un compte avec DynamoDB

DynamoDB AWS propose des points de terminaison basés sur des comptes qui peuvent améliorer les performances en utilisant AWS votre identifiant de compte pour rationaliser le routage des demandes.

Pour tirer parti de cette fonctionnalité, vous devez utiliser la version 3.656.0 ou supérieure de la AWS SDK pour JavaScript version 3. Cette fonctionnalité de point de terminaison basée sur un compte est activée par défaut dans cette nouvelle version.

Si vous souhaitez désactiver le routage basé sur le compte, vous disposez des options suivantes :

- Configurez un client de service DynamoDB avec `accountIdEndpointMode` le paramètre défini sur. `disabled`
- Définissez la variable d'environnement `AWS_ACCOUNT_ID_ENDPOINT_MODE` sur`disabled`.
- Mettez à jour le paramètre du fichier de AWS configuration partagé `account_id_endpoint_mode` sur`disabled`.

L'extrait suivant illustre comment désactiver le routage basé sur un compte en configurant un client de service DynamoDB :

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

Le guide de référence AWS SDKs and Tools fournit plus d'informations sur les autres options de configuration.

Protection de l'intégrité des données avec les checksums d'Amazon S3

Amazon Simple Storage Service (Amazon S3) permet de spécifier une somme de contrôle lorsque vous chargez un objet. Lorsque vous spécifiez une somme de contrôle, elle est stockée avec l'objet et peut être validée lors du téléchargement de l'objet.

Les checksums fournissent une couche supplémentaire d'intégrité des données lorsque vous transférez des fichiers. Avec les checksums, vous pouvez vérifier la cohérence des données en confirmant que le fichier reçu correspond au fichier d'origine. Pour plus d'informations sur les checksums avec Amazon S3, consultez le guide de l'utilisateur d'Amazon Simple Storage Service, y compris les algorithmes pris en charge.

Vous avez la possibilité de choisir l'algorithme qui répond le mieux à vos besoins et de laisser le SDK calculer le checksum. Vous pouvez également fournir une valeur de somme de contrôle précalculée à l'aide de l'un des algorithmes pris en charge.

> **Note**
>
> À partir de la version 3.729.0 du AWS SDK pour JavaScript, le SDK fournit des protections d'intégrité par défaut en calculant automatiquement une `CRC32` somme de contrôle pour les téléchargements. Le SDK calcule cette somme de contrôle si vous ne fournissez pas de valeur de somme de contrôle précalculée ou si vous ne spécifiez pas d'algorithme que le SDK doit utiliser pour calculer une somme de contrôle.
> Le SDK fournit également des paramètres globaux pour les protections de l'intégrité des données que vous pouvez définir en externe, que vous pouvez consulter dans le Guide de référence AWS SDKs et sur les outils.

Charger un objet

Vous chargez des objets sur Amazon S3 à l'aide de la PutObjectcommande du`S3Client`. Utilisez le `ChecksumAlgorithm` paramètre du générateur pour activer le calcul `PutObjectRequest` de la somme de contrôle et spécifier l'algorithme. Consultez les algorithmes de somme de contrôle pris en charge pour connaître les valeurs valides.

L'extrait de code suivant montre une demande de téléchargement d'un objet avec une somme de contrôle CRC-32. Lorsque le SDK envoie la demande, il calcule le checksum CRC-32 et télécharge l'objet. Amazon S3 stocke le checksum avec l'objet.

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

const client = new S3();
const response = await client.putObject({
  Bucket: "my-bucket",
```

```
  Key: "my-key",
  Body: "Hello, world!",
  ChecksumAlgorithm: ChecksumAlgorithm.CRC32,
});
```

Si vous ne fournissez pas d'algorithme de somme de contrôle avec la demande, le comportement de somme de contrôle varie en fonction de la version du SDK que vous utilisez, comme indiqué dans le tableau suivant.

Comportement de somme de contrôle lorsqu'aucun algorithme de somme de contrôle n'est fourni

SDK pour la version JavaScript	Comportement de Checksum
Antérieur à 3.729.0	Le SDK ne calcule pas automatiquement une somme de contrôle basée sur le CRC et ne la fournit pas dans la demande.
3.729.0 ou version ultérieure	Le SDK utilise l'`CRC32`algorithme pour calculer le checksum et le fournit dans la demande. Amazon S3 valide l'intégrité du transfert en calculant sa propre somme de `CRC32` contrôle et en la comparant à la somme de contrôle fournie par le SDK. Si les sommes de contrôle correspondent, la somme de contrôle est enregistrée avec l'objet.

Si la somme de contrôle calculée par le SDK ne correspond pas à la somme de contrôle calculée par Amazon S3 lorsqu'il reçoit la demande, une erreur est renvoyée.

Utiliser une valeur de somme de contrôle précalculée

Une valeur de somme de contrôle précalculée fournie avec la demande désactive le calcul automatique par le SDK et utilise la valeur fournie à la place.

L'exemple suivant montre une demande avec une somme de contrôle SHA-256 précalculée.

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

```
const client = new S3();

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

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

Si Amazon S3 détermine que la valeur de la somme de contrôle est incorrecte pour l'algorithme spécifié, le service renvoie une réponse d'erreur.

Chargements partitionnés

Vous pouvez également utiliser des checksums pour les téléchargements partitionnés. Ils AWS SDK pour JavaScript peuvent utiliser les options de la `Upload` bibliothèque pour former et `@aws-sdk/lib-storage` utiliser des checksums avec des téléchargements partitionnés.

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

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

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

SDK pour les exemples de JavaScript code

Les rubriques de cette section contiennent des exemples d'utilisation AWS SDK pour JavaScript APIs de différents services pour effectuer des tâches courantes.

Trouvez le code source de ces exemples et d'autres dans le référentiel d'exemples de AWS code sur GitHub. Pour proposer un nouvel exemple de code que l'équipe de AWS documentation pourrait envisager de produire, créez une demande. L'équipe cherche à produire des exemples de code qui couvrent des scénarios et des cas d'utilisation plus larges, plutôt que de simples extraits de code qui couvrent uniquement les appels d'API individuels. Pour obtenir des instructions, consultez la section Code de création dans les directives de contribution sur GitHub.

Important

Ces exemples utilisent la syntaxe d' ECMAScript6 import/export.

- Cela nécessite la version 14.17 ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
- Si vous préférez utiliser la syntaxe CommonJS, consultez les directives JavaScript ES6Syntaxe /CommonJS de conversion.

Rubriques

- Création d'événements planifiés pour exécuter AWS Lambda des fonctions
- Création d’un chatbot Amazon Lex

JavaScript ES6Syntaxe /CommonJS

Les exemples de AWS SDK pour JavaScript code sont écrits en ECMAScript 6 (ES6). ES6 apporte une nouvelle syntaxe et de nouvelles fonctionnalités pour rendre votre code plus moderne et lisible, et faire plus encore.

ES6 nécessite que vous utilisiez Node.js version 13.x ou supérieure. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. Toutefois, si vous préférez, vous pouvez convertir n'importe lequel de nos exemples en syntaxe CommonJS en suivant les directives suivantes :

- `"type" : "module"`Supprimez-le de `package.json` l'environnement de votre projet.
- Convertissez toutes les ES6 `import` instructions en instructions CommonJS`require`. Par exemple, convertissez :

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

  À son équivalent CommonJS :

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

- Convertissez toutes les ES6 `export` instructions en instructions CommonJS`module.exports`. Par exemple, convertissez :

  ```
  export {s3}
  ```

  À son équivalent CommonJS :

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

L'exemple suivant illustre l'exemple de code permettant de créer un compartiment Amazon S3 à la fois dans CommonJS ES6 et dans CommonJS.

ES6

libs/s3Client.js

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS region
const REGION = "eu-west-1"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
export {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using ES6 syntax.
 import { CreateBucketCommand } from "@aws-sdk/client-s3";
 import { s3 } from "./libs/s3Client.js";

// Get service clients module and commands using CommonJS syntax.
// const { CreateBucketCommand } = require("@aws-sdk/client-s3");
// const { s3 } = require("./libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

CommonJS

libs/s3Client.js

```
// Create service client module using CommonJS syntax.
 const { S3Client } = require("@aws-sdk/client-s3");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
 // Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
 module.exports ={s3};
```

s3_createbucket.js

```
// Get service clients module and commands using CommonJS syntax.
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("./libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

AWS Elemental MediaConvert exemples

AWS Elemental MediaConvert est un service de transcodage vidéo basé sur des fichiers doté de fonctionnalités adaptées à la diffusion. Vous pouvez l'utiliser pour créer des ressources destinées à la diffusion et à la diffusion video-on-demand (VOD) sur Internet. Pour plus d'informations, consultez le Guide de l'utilisateur AWS Elemental MediaConvert .

L' JavaScript API pour MediaConvert est exposée via la classe `MediaConvert` client. Pour plus d'informations, consultez Class : MediaConvert dans la référence de l'API.

Rubriques

- Création et gestion de tâches de transcodage dans MediaConvert
- Utilisation de modèles de tâches dans MediaConvert

Création et gestion de tâches de transcodage dans MediaConvert

Cet exemple de code Node.js présente :

- Comment créer des tâches de transcodage dans. MediaConvert
- Procédure d'annulation d'une tâche de transcodage.
- Procédure de récupération de l'objet JSON pour une tâche de transcodage terminée.
- Procédure de récupération d'un tableau JSON pour jusqu'à 20 tâches créées en dernier.

Le scénario

Dans cet exemple, vous utilisez un module Node.js à appeler pour MediaConvert créer et gérer des tâches de transcodage. Le code utilise le SDK pour ce JavaScript faire en utilisant les méthodes suivantes de la classe MediaConvert client :

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

- `ListJobsCommand`

Tâches préalables

Pour configurer et exécuter cet exemple, réalisez tout d'abord les tâches ci-après :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions indiquées sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.
- Créez et configurez des compartiments Amazon S3 qui fournissent du stockage pour les fichiers d'entrée et de sortie des tâches. Pour plus de détails, voir Création d'un espace de stockage pour les fichiers dans le Guide de AWS Elemental MediaConvert l'utilisateur.
- Téléchargez la vidéo d'entrée dans le compartiment Amazon S3 que vous avez configuré pour le stockage d'entrée. Pour obtenir la liste des codecs et conteneurs vidéo d'entrée pris en charge, consultez la section Codecs et conteneurs d'entrée pris en charge dans le guide de l'utilisateur.AWS Elemental MediaConvert
- Créez un rôle IAM qui donne MediaConvert accès à vos fichiers d'entrée et aux compartiments Amazon S3 dans lesquels vos fichiers de sortie sont stockés. Pour plus de détails, consultez la section Configurer les autorisations IAM dans le guide de l'AWS Elemental MediaConvert utilisateur.

> **Important**
>
> Cet exemple utilise ECMAScript6 (ES6). Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> Toutefois, si vous préférez utiliser la syntaxe CommonJS, veuillez vous référer àJavaScript ES6Syntaxe /CommonJS.

Définition d'une tâche de transcodage simple

Créez un module Node.js nommé `emc_createjob.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez le code JSON qui définit les paramètres de tâche de transcodage.

Ces paramètres sont assez détaillés. Vous pouvez utiliser la AWS Elemental MediaConvert console pour générer les paramètres de tâche JSON en choisissant vos paramètres de tâche dans la console, puis en choisissant Afficher le JSON de la tâche en bas de la section Job. Cet exemple illustre le code JSON pour une tâche simple.

> **Note**
>
> *JOB_QUEUE_ARN*Remplacez-le par la file d'attente *IAM_ROLE_ARN* des MediaConvert tâches, par le nom de ressource Amazon (ARN) du rôle IAM, *OUTPUT_BUCKET_NAME* par le nom du compartiment de destination, par exemple, « s3://OUTPUT_BUCKET_NAME/ », et *INPUT_BUCKET_AND_FILENAME* par le compartiment d'entrée et le nom du fichier, par exemple, « s3://INPUT_BUCKET/FILE_NAME ».

```
const params = {
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN", //IAM_ROLE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "OUTPUT_BUCKET_NAME", //OUTPUT_BUCKET_NAME, e.g., "s3://
BUCKET_NAME/"
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
```

```
            TimecodeInsertion: "DISABLED",
            AntiAlias: "ENABLED",
            Sharpness: 50,
            CodecSettings: {
              Codec: "H_264",
              H264Settings: {
                InterlaceMode: "PROGRESSIVE",
                NumberReferenceFrames: 3,
                Syntax: "DEFAULT",
                Softness: 0,
                GopClosedCadence: 1,
                GopSize: 90,
                Slices: 1,
                GopBReference: "DISABLED",
                SlowPal: "DISABLED",
                SpatialAdaptiveQuantization: "ENABLED",
                TemporalAdaptiveQuantization: "ENABLED",
                FlickerAdaptiveQuantization: "DISABLED",
                EntropyEncoding: "CABAC",
                Bitrate: 5000000,
                FramerateControl: "SPECIFIED",
                RateControlMode: "CBR",
                CodecProfile: "MAIN",
                Telecine: "NONE",
                MinIInterval: 0,
                AdaptiveQuantization: "HIGH",
                CodecLevel: "AUTO",
                FieldEncoding: "PAFF",
                SceneChangeDetect: "ENABLED",
                QualityTuningLevel: "SINGLE_PASS",
                FramerateConversionAlgorithm: "DUPLICATE_DROP",
                UnregisteredSeiTimecode: "DISABLED",
                GopSizeUnits: "FRAMES",
                ParControl: "SPECIFIED",
                NumberBFramesBetweenReferenceFrames: 2,
                RepeatPps: "DISABLED",
                FramerateNumerator: 30,
                FramerateDenominator: 1,
                ParNumerator: 1,
                ParDenominator: 1,
              },
            },
            AfdSignaling: "NONE",
            DropFrameTimecode: "ENABLED",
```

```
            RespondToAfd: "NONE",
            ColorMetadata: "INSERT",
          },
          AudioDescriptions: [
            {
              AudioTypeControl: "FOLLOW_INPUT",
              CodecSettings: {
                Codec: "AAC",
                AacSettings: {
                  AudioDescriptionBroadcasterMix: "NORMAL",
                  RateControlMode: "CBR",
                  CodecProfile: "LC",
                  CodingMode: "CODING_MODE_2_0",
                  RawFormat: "NONE",
                  SampleRate: 48000,
                  Specification: "MPEG4",
                  Bitrate: 64000,
                },
              },
              LanguageCodeControl: "FOLLOW_INPUT",
              AudioSourceName: "Audio Selector 1",
            },
          ],
          ContainerSettings: {
            Container: "MP4",
            Mp4Settings: {
              CslgAtom: "INCLUDE",
              FreeSpaceBox: "EXCLUDE",
              MoovPlacement: "PROGRESSIVE_DOWNLOAD",
            },
          },
          NameModifier: "_1",
        },
      ],
    },
  ],
  AdAvailOffset: 0,
  Inputs: [
    {
      AudioSelectors: {
        "Audio Selector 1": {
          Offset: 0,
          DefaultSelection: "NOT_DEFAULT",
          ProgramSelection: 1,
```

```
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
 "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};
```

Création d'une tâche de transcodage

Après avoir créé les paramètres de tâche au format JSON, appelez la `run` méthode asynchrone pour appeler un objet de service `MediaConvert` client en transmettant les paramètres. L'ID de la tâche créée est renvoyé dans les données `data` de la réponse.

```
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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node emc_createjob.js
```

Cet exemple de code complet se trouve ici GitHub.

Annulation d'une tâche de transcodage

Créez un module Node.js nommé `emc_canceljob.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Créez l'objet JSON qui inclut l'ID de la tâche à annuler. Appelez ensuite la `CancelJobCommand` méthode en créant une promesse pour invoquer un objet de service `MediaConvert` client, en transmettant les paramètres. Traitez la réponse dans le rappel de promesse.

> **Note**
>
> Remplacez *`JOB_ID`* par l'ID de la tâche à annuler.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node ec2_canceljob.js
```

Cet exemple de code se trouve ici GitHub.

Liste des travaux de transcodage récents

Créez un module Node.js nommé `emc_listjobs.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez les paramètres JSON, y compris des valeurs indiquant s'il faut trier la liste en `ASCENDING` fonction ou en `DESCENDING` ordre, de l'Amazon Resource Name (ARN) de la file d'attente de tâches à vérifier et du statut des tâches à inclure. Appelez ensuite la `ListJobsCommand` méthode en créant une promesse pour invoquer un objet de service `MediaConvert` client, en transmettant les paramètres.

> **Note**
>
> *`QUEUE_ARN`*Remplacez-le par le Amazon Resource Name (ARN) de la file d'attente de tâches à vérifier et *`STATUS`* par le statut de la file d'attente.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node emc_listjobs.js
```

Cet exemple de code se trouve ici GitHub.

Utilisation de modèles de tâches dans MediaConvert

Cet exemple de code Node.js présente :

- Comment créer des modèles de AWS Elemental MediaConvert tâches.
- Procédure d'utilisation d'un modèle de tâche pour créer une tâche de transcodage.
- Procédure permettant de répertorier tous vos modèles de tâche.
- Procédure de suppression des modèles de tâche.

Le scénario

Le JSON requis pour créer une tâche de transcodage dans MediaConvert est détaillé et contient un grand nombre de paramètres. Vous pouvez simplifier considérablement la création des tâches en enregistrant les paramètres que vous savez appropriés dans un modèle de tâche, que vous utiliserez par la suite pour créer d'autres tâches. Dans cet exemple, vous utilisez un module Node.js à appeler MediaConvert pour créer, utiliser et gérer des modèles de tâches. Le code utilise le SDK pour ce JavaScript faire en utilisant les méthodes suivantes de la classe MediaConvert client :

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

Tâches préalables

Pour configurer et exécuter cet exemple, réalisez tout d'abord les tâches ci-après :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions indiquées sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.
- Créez un rôle IAM qui donne MediaConvert accès à vos fichiers d'entrée et aux compartiments Amazon S3 dans lesquels vos fichiers de sortie sont stockés. Pour plus de détails, consultez la section Configurer les autorisations IAM dans le guide de l'AWS Elemental MediaConvert utilisateur.

> **Important**
>
> Ces exemples utilisent ECMAScript6 (ES6). Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> Toutefois, si vous préférez utiliser la syntaxe CommonJS, veuillez vous référer àJavaScript ES6Syntaxe /CommonJS.

Création d'un modèle de tâche

Créez un module Node.js nommé `emc_create_jobtemplate.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Spécifiez l'objet JSON des paramètres pour la création du modèle. Vous pouvez utiliser la plupart des paramètres JSON issus d'une tâche antérieure réussie afin de spécifier les valeurs `Settings` du modèle. Cet exemple utilise les paramètres de tâche de Création et gestion de tâches de transcodage dans MediaConvert.

Appelez la `CreateJobTemplateCommand` méthode en créant une promesse pour appeler un objet de service `MediaConvert` client, en transmettant les paramètres.

> **Note**
>
> Remplacez-le *JOB_QUEUE_ARN* par le nom de ressource Amazon (ARN) de la file d'attente des tâches à vérifier et *BUCKET_NAME* par le nom du compartiment Amazon S3 de destination, par exemple « 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node emc_create_jobtemplate.js
```

Cet exemple de code se trouve ici GitHub.

Création d'une tâche de transcodage à partir d'un modèle de tâche

Créez un module Node.js nommé `emc_template_createjob.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez l'objet JSON des paramètres de création de tâche, en incluant le nom du modèle de tâche à utiliser et les `Settings` à utiliser propres à la tâche que vous créez. Appelez ensuite la `CreateJobsCommand` méthode en créant une promesse pour invoquer un objet de service `MediaConvert` client, en transmettant les paramètres.

> **Note**
>
> *JOB_QUEUE_ARN*Remplacez-le par le nom de ressource Amazon (ARN) de la file d'attente de tâches pour vérifier, *KEY_PAIR_NAME TEMPLATE_NAME* avec, *ROLE_ARN* avec le nom de ressource Amazon (ARN) du rôle, ainsi *INPUT_BUCKET_AND_FILENAME* que

par le compartiment d'entrée et le nom du fichier, par exemple, « s3://BUCKET_NAME/FILE_NAME ».

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

const params = {
  Queue: "QUEUE_ARN", //QUEUE_ARN
  JobTemplate: "TEMPLATE_NAME", //TEMPLATE_NAME
  Role: "ROLE_ARN", //ROLE_ARN
  Settings: {
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
 "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
  },
};

const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
```

```
    console.log("Success! ", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node emc_template_createjob.js
```

Cet exemple de code se trouve ici GitHub.

Répertorier vos modèles de tâches

Créez un module Node.js nommé `emc_listtemplates.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet afin de transmettre les paramètres de demande pour la méthode `listTemplates` de la classe client `MediaConvert`. Incluez les valeurs permettant de déterminer quels modèles répertorier (`NAME`, `CREATION_DATE`, `SYSTEM`), combien de modèles répertorier et leur ordre de tri. Pour appeler la `ListTemplatesCommand` méthode, créez une promesse pour appeler un objet de service MediaConvert client en transmettant les paramètres.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node emc_listtemplates.js
```

Cet exemple de code se trouve ici GitHub.

Supprimer un modèle de tâche

Créez un module Node.js nommé `emc_deletetemplate.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet qui permettra de transmettre le nom du modèle de tâche que vous souhaitez supprimer en tant que paramètres de la méthode `DeleteJobTemplateCommand` de la classe client `MediaConvert`. Pour appeler la `DeleteJobTemplateCommand` méthode, créez une promesse pour appeler un objet de service MediaConvert client en transmettant les paramètres.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node emc_deletetemplate.js
```

Cet exemple de code se trouve ici GitHub.

AWS Lambda exemples

AWS Lambda est un service de calcul sans serveur qui vous permet d'exécuter du code sans provisionner ni gérer de serveurs, créer une logique de dimensionnement des clusters adaptée à la charge de travail, gérer les intégrations d'événements ou gérer les temps d'exécution.

L' JavaScript API pour AWS Lambda est exposée via la classe LambdaServiceclient.

Voici une liste d'exemples qui montrent comment créer et utiliser des fonctions Lambda avec la AWS SDK pour JavaScript version 3 :

- Invoquer Lambda avec API Gateway
- Création d'événements planifiés pour exécuter AWS Lambda des fonctions

Exemples Amazon Lex

Amazon Lex est un AWS service permettant de créer des interfaces conversationnelles dans des applications utilisant la voix et le texte.

L' JavaScript API d'Amazon Lex est exposée via la classe client Lex Runtime Service.

- Création d’un chatbot Amazon Lex

Exemples d'Amazon Polly

Cet exemple de code Node.js présente :

- Importer du contenu audio enregistré à l'aide d'Amazon Polly vers Amazon S3

Le scénario

Dans cet exemple, une série de modules Node.js sont utilisés pour télécharger automatiquement le son enregistré à l'aide d'Amazon Polly vers Amazon S3 en utilisant les méthodes suivantes de la classe client Amazon S3 :

- `StartSpeechSynthesisTaskCommand`

Tâches préalables

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez un environnement de projet pour exécuter JavaScript des exemples de nœuds en suivant les instructions figurant sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.
- Créez une enquête de rôle d'utilisateur Amazon Cognito non authentifiée Gestion des identités et des accès AWS (IAM) : autorisations, SynthesizeSpeech et un pool d'identités Amazon Cognito auquel le rôle IAM est associé. La Créez les AWS ressources à l'aide du CloudFormation section ci-dessous décrit comment créer ces ressources.

> **Note**
>
> Cet exemple utilise Amazon Cognito, mais si vous n'utilisez pas Amazon Cognito, AWS votre utilisateur doit avoir la politique d'autorisation IAM suivante
>
> JSON

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

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

Créez les AWS ressources à l'aide du CloudFormation

CloudFormation vous permet de créer et de provisionner des déploiements AWS d'infrastructure de manière prévisible et répétée. Pour plus d'informations CloudFormation, consultez le guide de AWS CloudFormation l'utilisateur.

Pour créer la CloudFormation pile :

1. Installez et configurez en AWS CLI suivant les instructions du guide de l'AWS CLI utilisateur.
2. Créez un fichier nommé `setup.yaml` dans le répertoire racine du dossier de votre projet et GitHubcopiez-y le contenu.

> **Note**
>
> Le CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour plus d'informations à ce sujet AWS CDK, consultez le guide du AWS Cloud Development Kit (AWS CDK) développeur.

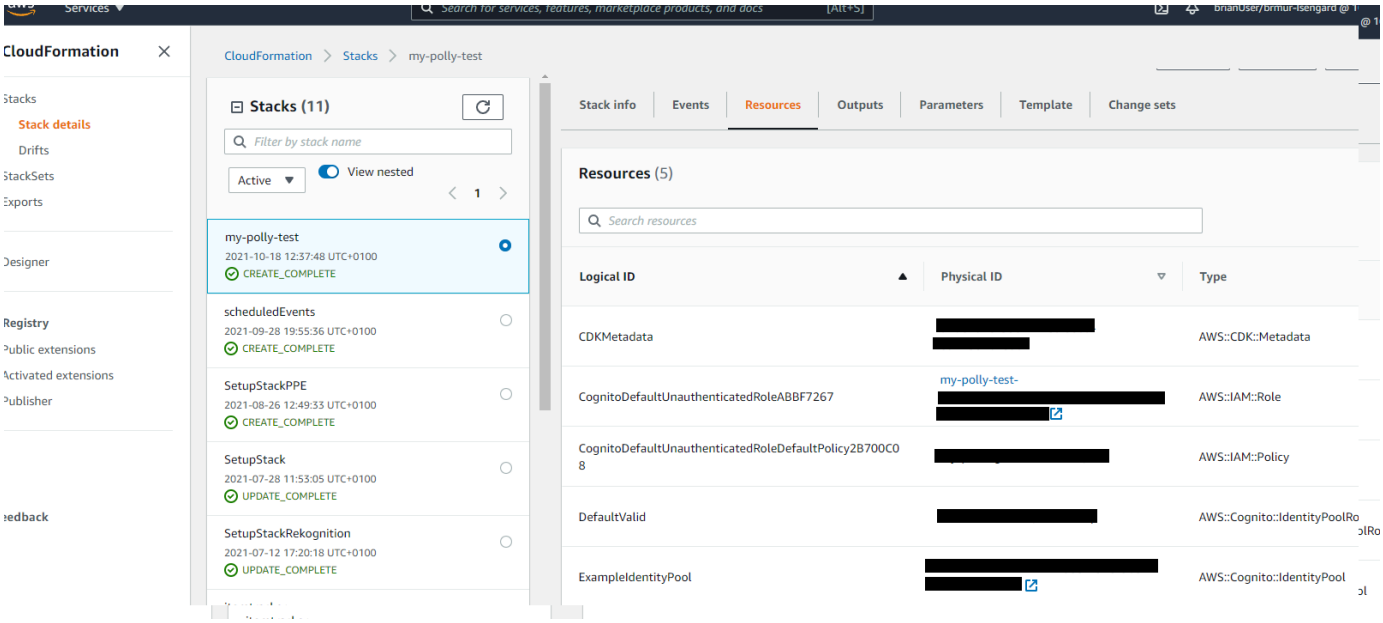
3. Exécutez la commande suivante depuis la ligne de commande, en la *`STACK_NAME`* remplaçant par un nom unique pour la pile.

> **Important**
>
> Le nom de la pile doit être unique au sein d'une AWS région et d'un AWS compte. Vous pouvez spécifier jusqu'à 128 caractères. Les chiffres et les tirets sont autorisés.

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

Pour plus d'informations sur les paramètres de `create-stack` commande, consultez le guide de référence des AWS CLI commandes et le guide de CloudFormation l'utilisateur.

4. Accédez à la console CloudFormation de gestion, choisissez Stacks, choisissez le nom de la pile, puis cliquez sur l'onglet Ressources pour afficher la liste des ressources créées.



Importer du contenu audio enregistré à l'aide d'Amazon Polly vers Amazon S3

Créez un module Node.js nommé `polly_synthesize_to_s3.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Dans le code, entrez le*REGION*, et le*BUCKET_NAME*. Pour accéder à Amazon Polly, créez un objet de service `Polly` client. *"IDENTITY_POOL_ID"*Remplacez-le par `IdentityPoolId` celui de la page d'exemple du pool d'identités Amazon Cognito que vous avez créé pour cet exemple. Ceci est également transmis à chaque objet client.

Appelez la `StartSpeechSynthesisCommand` méthode de l'objet du service client Amazon Polly, synthétisez le message vocal et chargez-le dans le compartiment Amazon S3.

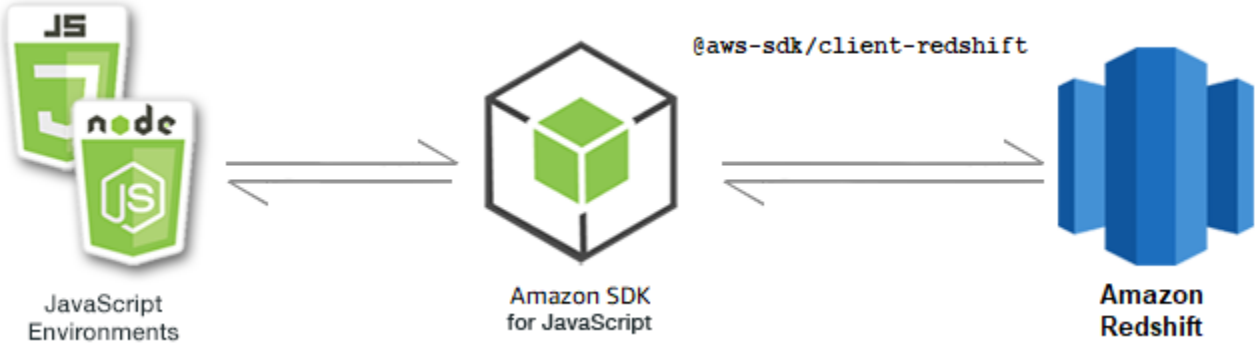
```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "./libs/pollyClient.js";
```

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

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

Cet exemple de code se trouve ici sur GitHub.

Exemples d'Amazon Redshift

Amazon Redshift est un service d'entrepôt des données entièrement géré dans le cloud. Un entrepôt des données Amazon Redshift est un ensemble de ressources informatiques appelées nœuds, qui sont organisées en un groupe appelé cluster. Chaque cluster exécute un moteur Amazon Redshift et contient une ou plusieurs bases de données.



L' JavaScript API d'Amazon Redshift est exposée via la classe client Amazon Redshift.

Rubriques

- Exemples d'Amazon Redshift

Exemples d'Amazon Redshift

Dans cet exemple, une série de modules Node.js sont utilisés pour créer, modifier, décrire les paramètres des clusters Amazon Redshift, puis les supprimer à l'aide des méthodes suivantes de la classe `Redshift` client :

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

Pour plus d'informations sur les utilisateurs d'Amazon Redshift, consultez le guide de démarrage d'Amazon Redshift.

Tâches préalables

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions indiquées sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets de service client et des commandes à l'aide de ECMAScript6 ()ES6.
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .

- Si vous préférez utiliser la syntaxe CommonJS, voir JavaScript ES6Syntaxe /CommonJS

Création d'un cluster Amazon Redshift

Cet exemple montre comment créer un cluster Amazon Redshift à l'aide du. AWS SDK pour JavaScript Pour de plus amples informations, veuillez consulter CreateCluster.

> **Important**
>
> Le cluster que vous êtes sur le point de créer est actif (et ne fonctionne pas dans un sandbox). Des frais d'utilisation sont perçus pour l'utilisation d'Amazon Redshift pour le cluster jusqu'à ce que vous le supprimiez. Si vous supprimez le cluster au cours de la même séance que lors de sa création, les frais totaux sont minimes.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`redshiftClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Redshift. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `redshift-create-cluster.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez un objet de paramètres, en spécifiant le type de nœud à provisionner, les informations d'identification de connexion principale pour l'instance de base de données créée automatiquement dans le cluster, et enfin le type de cluster.

> **Note**
>
> Remplacez *`CLUSTER_NAME`* par le nom du cluster. Pour *`NODE_TYPE`* spécifier le type de nœud à provisionner, tel que « dc2.large », par exemple. *`MASTER_USERNAME`*et

MASTER_USER_PASSWORD sont les informations d'identification de connexion de l'utilisateur principal de votre instance de base de données dans le cluster. Pour*CLUSTER_TYPE*, entrez le type de cluster. Si vous le spécifiez`single-node`, vous n'avez pas besoin du `NumberOfNodes` paramètre. Les autres paramètres sont facultatifs.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Modifier un cluster Amazon Redshift

Cet exemple montre comment modifier le mot de passe utilisateur principal d'un cluster Amazon Redshift à l'aide du. AWS SDK pour JavaScript Pour plus d'informations sur les autres paramètres que vous pouvez modifier, consultez ModifyCluster.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`redshiftClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Redshift. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `redshift-modify-cluster.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Spécifiez la AWS région, le nom du cluster que vous souhaitez modifier et le nouveau mot de passe de l'utilisateur principal.

> **Note**
>
> *`CLUSTER_NAME`*Remplacez-le par le nom du cluster et *`MASTER_USER_PASSWORD`* par le nouveau mot de passe de l'utilisateur principal.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Afficher les détails d'un cluster Amazon Redshift

Cet exemple montre comment afficher les détails d'un cluster Amazon Redshift à l'aide du. AWS SDK pour JavaScript Pour plus d'informations sur les options facultatives, consultez DescribeClusters.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`redshiftClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Redshift. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `redshift-describe-clusters.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages

requis. Spécifiez la AWS région, le nom du cluster que vous souhaitez modifier et le nouveau mot de passe de l'utilisateur principal.

> **Note**
>
> Remplacez *CLUSTER_NAME* par le nom du cluster.

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

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

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

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Supprimer un cluster Amazon Redshift

Cet exemple montre comment afficher les détails d'un cluster Amazon Redshift à l'aide du. AWS SDK pour JavaScript Pour plus d'informations sur les autres paramètres que vous pouvez modifier, consultez DeleteCluster.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`redshiftClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Redshift. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js avec le nom du fichier`redshift-delete-clusters.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Spécifiez la AWS région, le nom du cluster que vous souhaitez modifier et le nouveau mot de passe de l'utilisateur principal. Spécifiez ensuite si vous souhaitez enregistrer un instantané final du cluster avant de le supprimer et, dans l'affirmative, l'ID de l'instantané.

> **Note**
>
> Remplacez *`CLUSTER_NAME`* par le nom du cluster. Pour le*`SkipFinalClusterSnapshot`*, spécifiez s'il faut créer un instantané final du cluster avant de le supprimer. Si vous spécifiez « faux », spécifiez l'identifiant du cliché final du cluster dans*`CLUSTER_SNAPSHOT_ID`*. Vous pouvez obtenir cet identifiant en cliquant sur le lien dans la colonne Snapshots du cluster sur le tableau de bord des clusters, puis en faisant défiler la page vers le bas jusqu'au volet Snapshots. Notez que le radical ne `rs:` fait pas partie de l'identifiant du cliché.

```
// 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();
```

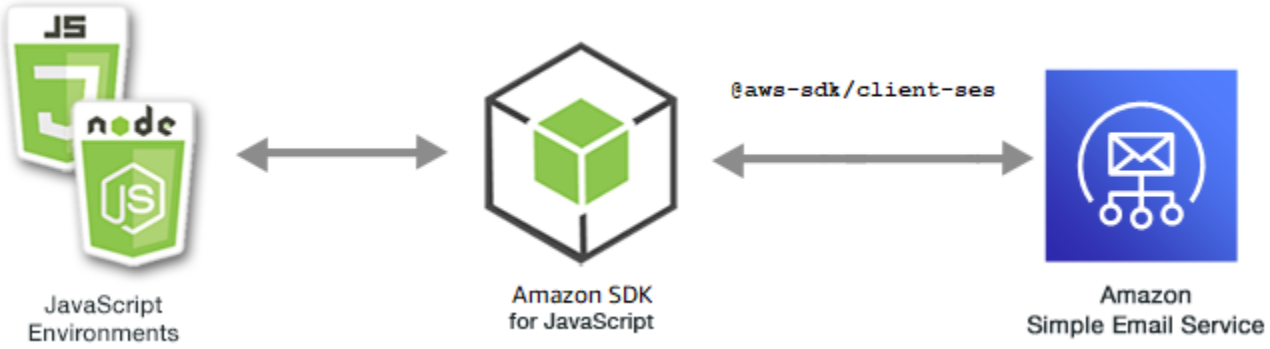
Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Exemples d'Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) est un service d'envoi d'e-mails basé sur le cloud conçu pour aider les spécialistes du marketing numérique et les développeurs d'applications à envoyer des e-mails marketing, de notification et transactionnels. C'est un service fiable et rentable pour les entreprises de toutes tailles utilisant les e-mails pour rester en contact avec leurs clients.



L' JavaScript API d'Amazon SES est exposée par le biais de la classe SES client. Pour plus d'informations sur l'utilisation de la classe client Amazon SES, consultez Classe : SES dans le Guide de référence des API.

Rubriques

- Gestion des identités Amazon SES
- Utilisation de modèles d'e-mail dans Amazon SES
- Envoi d'e-mails à l'aide d'Amazon SES

Gestion des identités Amazon SES

Cet exemple de code Node.js présente :

- Comment vérifier les adresses e-mail et les domaines utilisés avec Amazon SES.
- Comment attribuer une politique Gestion des identités et des accès AWS (IAM) à vos identités Amazon SES.
- Comment répertorier toutes les identités Amazon SES associées à votre AWS compte.
- Comment supprimer les identités utilisées avec Amazon SES

Une identité Amazon SES est une adresse e-mail ou un domaine qu'Amazon SES utilise pour envoyer des e-mails. Amazon SES vous demande de vérifier votre identité e-mail, de confirmer que vous en êtes le propriétaire et d'empêcher les autres de les utiliser.

Pour en savoir plus sur la façon de vérifier les adresses e-mail et les domaines dans Amazon SES, consultez la section Vérification des adresses e-mail et des domaines dans Amazon SES dans le manuel Amazon Simple Email Service Developer Guide. Pour plus d'informations sur l'autorisation d'envoi dans Amazon SES, consultez Présentation de l'autorisation d'envoi Amazon SES.

Le scénario

Dans cet exemple, vous utilisez une série de modules Node.js pour vérifier et gérer les identités Amazon SES. Les modules Node.js utilisent le SDK JavaScript pour vérifier les adresses e-mail et les domaines, en utilisant les méthodes suivantes de la classe SES client :

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

Tâches préalables

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions figurant sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets de service client et des commandes à l'aide de ECMAScript6 ()ES6.
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> - Si vous préférez utiliser la syntaxe CommonJS, consultezJavaScript ES6Syntaxe / CommonJS.

Répertorier vos identités

Dans cet exemple, utilisez un module Node.js pour répertorier les adresses e-mail et les domaines à utiliser avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_listidentities.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre le paramètre `IdentityType` ainsi que les autres paramètres de la méthode `ListIdentitiesCommand` de la classe client `SES`. Pour appeler la `ListIdentitiesCommand` méthode, appelez un objet de service Amazon SES en transmettant l'objet de paramètres.

Le `data` résultat contient un tableau d'identités de domaine tel que spécifié par le `IdentityType` paramètre.

> **Note**
>
> Remplacez *`IdentityType`* par le type d'identité, qui peut être EmailAddress « » ou « Domaine ».

```
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;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node ses_listidentities.js
```

Cet exemple de code se trouve ici GitHub.

Vérification d'une identité d'adresse e-mail

Dans cet exemple, utilisez un module Node.js pour vérifier les expéditeurs d'e-mails à utiliser avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_verifyemailidentity.js`. Configurez le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis.

Créez un objet pour transmettre le paramètre `EmailAddress` ainsi que les autres paramètres de la méthode `VerifyEmailIdentityCommand` de la classe client SES. Pour appeler la `VerifyEmailIdentityCommand` méthode, appelez un objet du service client Amazon SES en transmettant les paramètres.

> **Note**
>
> Remplacez *`EMAIL_ADDRESS`* par l'adresse e-mail, telle que name@example.com.

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

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
```

```
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. Le domaine est ajouté à Amazon SES pour être vérifié.

```
node ses_verifyemailidentity.js
```

Cet exemple de code se trouve ici GitHub.

Vérification de l'identité d'un domaine

Dans cet exemple, utilisez un module Node.js pour vérifier les domaines de messagerie à utiliser avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_verifydomainidentity.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre le paramètre `Domain` ainsi que les autres paramètres de la méthode `VerifyDomainIdentityCommand` de la classe client `SES`. Pour appeler la `VerifyDomainIdentityCommand` méthode, appelez un objet du service client Amazon SES en transmettant l'objet de paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
> Remplacez *DOMAIN_NAME* par le nom de domaine.

```
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;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. Le domaine est ajouté à Amazon SES pour être vérifié.

```
node ses_verifydomainidentity.js
```

Cet exemple de code se trouve ici GitHub.

Supprimer des identités

Dans cet exemple, utilisez un module Node.js pour supprimer les adresses e-mail ou les domaines utilisés avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_deleteidentity.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre le paramètre `Identity` ainsi que les autres paramètres de la méthode `DeleteIdentityCommand` de la classe client `SES`. Pour appeler la `DeleteIdentityCommand` méthode, créez un `request` pour appeler un objet de service client Amazon SES en transmettant les paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
> Remplacez *IDENTITY_EMAIL* par l'e-mail de l'identité à supprimer.

```
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;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node ses_deleteidentity.js
```

Cet exemple de code se trouve ici GitHub.

Utilisation de modèles d'e-mail dans Amazon SES

Cet exemple de code Node.js présente :

- Comment obtenir une liste de tous vos modèles d'e-mails.
- Comment récupérer et mettre à jour des modèles d'e-mails.
- Comment créer et supprimer des modèles d'e-mails.

Amazon SES vous permet d'envoyer des e-mails personnalisés à l'aide de modèles d'e-mail. Pour en savoir plus sur la création et l'utilisation de modèles d'e-mails dans Amazon SES, consultez la section Envoi d'e-mails personnalisés à l'aide de l'API Amazon SES dans le manuel Amazon Simple Email Service Developer Guide.

Le scénario

Dans cet exemple, vous utilisez une série de modules Node.js à utiliser avec des modèles d'e-mail. Les modules Node.js utilisent le SDK pour JavaScript créer et utiliser des modèles de courrier électronique en utilisant les méthodes suivantes de la classe SES client :

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

Tâches préalables

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions figurant sur GitHub.

- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

Important

Ces exemples montrent comment importer/exporter des objets de service client et des commandes à l'aide de ECMAScript6 ()ES6.

- Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
- Si vous préférez utiliser la syntaxe CommonJS, consultezJavaScript ES6Syntaxe / CommonJS.

Répertorier vos modèles d'e-mails

Dans cet exemple, utilisez un module Node.js pour créer un modèle d'e-mail à utiliser avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_listtemplates.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre les paramètres de la méthode `ListTemplatesCommand` de la classe client `SES`. Pour appeler la `ListTemplatesCommand` méthode, appelez un objet du service client Amazon SES en transmettant les paramètres.

Note

Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple

à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

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

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. Amazon SES renvoie la liste des modèles.

```
node ses_listtemplates.js
```

Cet exemple de code se trouve ici GitHub.

Obtenir un modèle d'e-mail

Dans cet exemple, utilisez un module Node.js pour obtenir un modèle d'e-mail à utiliser avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_gettemplate.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre le paramètre `TemplateName` ainsi que les autres paramètres de la méthode `GetTemplateCommand` de la classe client `SES`. Pour appeler la `GetTemplateCommand` méthode, appelez un objet du service client Amazon SES en transmettant les paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
> Remplacez *`TEMPLATE_NAME`* par le nom du modèle à renvoyer.

```
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;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. Amazon SES renvoie les détails du modèle.

```
node ses_gettemplate.js
```

Cet exemple de code se trouve ici GitHub.

Création d'un modèle d'e-mail

Dans cet exemple, utilisez un module Node.js pour créer un modèle d'e-mail à utiliser avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_createtemplate.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre les paramètres de la méthode `CreateTemplateCommand` de la classe client `SES`, y compris `TemplateName`, `HtmlPart`, `SubjectPart` et `TextPart`. Pour appeler la `CreateTemplateCommand` méthode, appelez un objet du service client Amazon SES en transmettant les paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
> Remplacez-le *`TEMPLATE_NAME`* par le nom du nouveau modèle, *`HtmlPart`* par le contenu balisé HTML de l'e-mail et *`SubjectPart`* par l'objet de l'e-mail.

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

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

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

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

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

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. Le modèle est ajouté à Amazon SES.

```
node ses_createtemplate.js
```

Cet exemple de code se trouve ici GitHub.

Mise à jour d'un modèle d'e-mail

Dans cet exemple, utilisez un module Node.js pour créer un modèle d'e-mail à utiliser avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_updatetemplate.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre les valeurs de paramètre `Template` que vous souhaitez mettre à jour dans le modèle, avec le paramètre `TemplateName` obligatoire transmis à la méthode

`UpdateTemplateCommand` de la classe client `SES`. Pour appeler la `UpdateTemplateCommand` méthode, appelez un objet de service Amazon SES en transmettant les paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
*TEMPLATE_NAME*Remplacez-le par le nom du modèle et *HTML_PART* par le contenu HTML balisé de l'e-mail.

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

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

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

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

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
```

```
    console.log("Failed to update template.", err);
    return err;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. Amazon SES renvoie les détails du modèle.

```
node ses_updatetemplate.js
```

Cet exemple de code se trouve ici GitHub.

Suppression d'un modèle d'e-mail

Dans cet exemple, utilisez un module Node.js pour créer un modèle d'e-mail à utiliser avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_deletetemplate.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre le paramètre`TemplateName` obligatoire à la méthode `DeleteTemplateCommand` de la classe client `SES`. Pour appeler la `DeleteTemplateCommand` méthode, appelez un objet de service Amazon SES en transmettant les paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple

à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
> Remplacez *TEMPLATE_NAME* par le nom du modèle à supprimer.

```
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;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. Amazon SES renvoie les détails du modèle.

```
node ses_deletetemplate.js
```

Cet exemple de code se trouve ici GitHub.

Envoi d'e-mails à l'aide d'Amazon SES

Cet exemple de code Node.js présente :

- L'envoi d'un texte ou d'un e-mail au format HTML.
- L'envoi d'e-mails basés sur un modèle d'e-mail.
- L'envoi d'e-mails en bloc basés sur un modèle d'e-mail.

L'API Amazon SES vous permet d'envoyer un e-mail de deux manières différentes, en fonction du niveau de contrôle que vous souhaitez obtenir sur la composition du message : formaté et brut. Pour plus de détails, consultez Envoi d'e-mails formatés à l'aide de l'API Amazon SES et Envoi d'e-mails bruts à l'aide de l'API Amazon SES.

Le scénario

Dans cet exemple, vous utilisez une série de modules Node.js pour envoyer un e-mail de plusieurs façons. Les modules Node.js utilisent le SDK pour JavaScript créer et utiliser des modèles de courrier électronique en utilisant les méthodes suivantes de la classe SES client :

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

Tâches préalables

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions figurant sur GitHub.

- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez

la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets de service client et des commandes à l'aide de ECMAScript6 ()ES6.
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> - Si vous préférez utiliser la syntaxe CommonJS, consultezJavaScript ES6Syntaxe / CommonJS.

Exigences relatives à l'envoi de messages électroniques

Amazon SES rédige un e-mail et le met immédiatement en file d'attente pour envoi. Pour envoyer un e-mail à l'aide de la méthode `SendEmailCommand`, votre message doit répondre aux exigences suivantes :

- Vous devez envoyer le message à partir d'une adresse e-mail ou d'un domaine vérifié(e). Si vous essayez d'envoyer un e-mail à l'aide d'une adresse ou d'un domaine non vérifié(e), cela engendre une erreur `"Email address not verified"`.
- Si votre compte est encore dans l'environnement de test (sandbox) Amazon SES, vous pouvez uniquement envoyer un e-mail à des adresses ou des domaines vérifiés, ou à des adresses e-mail associées au simulateur de boîte de réception Amazon SES. Pour plus d'informations, consultez la section Vérification des adresses e-mail et des domaines dans le manuel Amazon Simple Email Service Developer Guide.
- La taille totale du message, pièces jointes comprises, doit être inférieure à 10 Mo.
- Le message doit inclure au moins un destinataire. L'adresse e-mail du destinataire peut se trouver dans le champ À :, Cc : ou Cci :. Si l'adresse e-mail d'un destinataire n'est pas valide (c'est-à-dire qu'elle n'est pas au format`UserName@[SubDomain.]Domain.TopLevelDomain`), le message entier est rejeté, même s'il contient d'autres destinataires valides.
- Le message ne peut pas inclure plus de 50 destinataires dans les champs To :, CC : et BCC :. Si vous avez besoin d'envoyer un e-mail à davantage de personnes, vous pouvez diviser votre liste

de destinataires en groupes de 50 ou moins, puis appeler la méthode `sendEmail` plusieurs fois pour envoyer le message à chaque groupe.

Envoi d'un e-mail

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_sendemail.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre les valeurs des paramètres qui définissent l'e-mail à envoyer, y compris les adresses de l'expéditeur et du destinataire, l'objet et le corps de l'e-mail au format texte brut et HTML, à la `SendEmailCommand` méthode de la classe `SES` client. Pour appeler la `SendEmailCommand` méthode, appelez un objet de service Amazon SES en transmettant les paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
> Remplacez *toAddress* par l'adresse à laquelle envoyer l'e-mail et *fromAddress* par l'adresse e-mail à partir de laquelle l'e-mail doit être envoyé.

```
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;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. L'e-mail est mis en file d'attente pour être envoyé par Amazon SES.

```
node ses_sendemail.js
```

Cet exemple de code se trouve ici GitHub.

Envoi d'un e-mail à l'aide d'un modèle

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES. Créez un module Node.js nommé `ses_sendtemplatedemail.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre les valeurs de paramètre qui définissent l'e-mail à envoyer, y compris l'expéditeur et le destinataire, l'objet, le corps du message en texte brut ou au format HTML, à la méthode `SendTemplatedEmailCommand` de la classe client `SES`. Pour appeler la `SendTemplatedEmailCommand` méthode, appelez un objet du service client Amazon SES en transmettant les paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
> Remplacez *`REGION`* par votre AWS région, *`USER`* par le nom et l'adresse e-mail auxquels envoyer l'e-mail, *`VERIFIED_EMAIL`* par l'adresse e-mail à partir de laquelle envoyer l'e-mail et *`TEMPLATE_NAME`* par le nom du modèle.

```
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;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. L'e-mail est mis en file d'attente pour être envoyé par Amazon SES.

```
node ses_sendtemplatedemail.js
```

Cet exemple de code se trouve ici GitHub.

Envoi d'e-mails en masse à l'aide d'un modèle

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`sesClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SES. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `ses_sendbulktemplatedemail.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre les valeurs des paramètres qui définissent l'e-mail à envoyer, y compris les adresses de l'expéditeur et du destinataire, l'objet et le corps de l'e-mail au format texte brut et HTML, à la `SendBulkTemplatedEmailCommand` méthode de la classe `SES` client. Pour appeler la `SendBulkTemplatedEmailCommand` méthode, appelez un objet de service Amazon SES en transmettant les paramètres.

> **Note**
>
> Cet exemple importe et utilise les clients du package AWS Service V3 requis, les commandes V3, et utilise la `send` méthode dans un modèle async/await. Vous pouvez créer cet exemple à l'aide des commandes V2 en apportant quelques modifications mineures. Pour plus de détails, consultez Utilisation des commandes v3.

> **Note**
>
> Remplacez-le *`USERS`* par les noms et adresses e-mail auxquels envoyer l'e-mail, *`VERIFIED_EMAIL_1`* par l'adresse e-mail à partir de laquelle l'e-mail doit être envoyé et *`TEMPLATE_NAME`* par le nom du modèle.

```
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;
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande. L'e-mail est mis en file d'attente pour être envoyé par Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Cet exemple de code se trouve ici GitHub.

Exemples du service de notification Amazon Simple

Amazon Simple Notification Service (Amazon SNS) est un service Web qui coordonne et gère la livraison ou l'envoi de messages aux points de terminaison ou aux clients abonnés.

Sur Amazon SNS, il existe deux types de clients, les éditeurs et les abonnés, également appelés producteurs et consommateurs.

Les éditeurs communiquent de façon asynchrone avec les abonnés en produisant et en envoyant un message à une rubrique, qui est un point d'accès logique et un canal de communication. Les abonnés (serveurs Web, adresses e-mail, files d'attente Amazon SQS, AWS Lambda fonctions) consomment ou reçoivent le message ou la notification via l'un des protocoles pris en charge (Amazon SQS, HTTP/S, e-mail, SMS AWS Lambda) lorsqu'ils sont abonnés au sujet.

L' JavaScript API d'Amazon SNS est exposée par le biais de la classe : SNS.

Rubriques

- Gestion des rubriques dans Amazon SNS
- Publication de messages sur Amazon SNS
- Gestion des abonnements sur Amazon SNS
- Envoi de SMS avec Amazon SNS

Gestion des rubriques dans Amazon SNS

Cet exemple de code Node.js présente :

- Comment créer des rubriques dans Amazon SNS sur lesquelles vous pouvez publier des notifications.
- Comment supprimer des sujets créés dans Amazon SNS.
- Comment obtenir une liste des rubriques disponibles.
- Comment obtenir et définir des attributs de rubrique.

Scénario

Dans cet exemple, vous utilisez une série de modules Node.js pour créer, répertorier et supprimer des rubriques Amazon SNS, ainsi que pour gérer les attributs des rubriques. Les modules Node.js utilisent le SDK pour gérer les sujets JavaScript à l'aide des méthodes suivantes de la classe SNS client :

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

Tâches prérequises

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions indiquées sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment utiliser des objets de service import/export client et des commandes en utilisant ECMAScript6 (ES6).
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> - Si vous préférez utiliser la syntaxe CommonJS, consultezJavaScript ES6Syntaxe / CommonJS.

Création d'une rubrique

Dans cet exemple, utilisez un module Node.js pour créer une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `create-topic.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet pour transmettre le paramètre`Name` de la nouvelle rubrique à la méthode `CreateTopicCommand` de la classe client SNS. Pour appeler la `CreateTopicCommand` méthode, créez une fonction asynchrone invoquant un objet de service Amazon SNS, en transmettant l'objet de paramètres. Le `data` résultat contient l'ARN du sujet.

> **Note**
>
> Remplacez *`TOPIC_NAME`* par le nom du sujet.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Liste de vos rubriques

Dans cet exemple, utilisez un module Node.js pour répertorier toutes les rubriques Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `list-topics.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet vide à transmettre à la méthode `ListTopicsCommand` de la classe client SNS. Pour appeler la `ListTopicsCommand` méthode, créez une fonction asynchrone invoquant un objet de

service Amazon SNS, en transmettant l'objet de paramètres. Le `data` fichier renvoyé contient un tableau de votre sujet Amazon Resource Names (ARNs).

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

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Suppression d'une rubrique

Dans cet exemple, utilisez un module Node.js pour supprimer une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `delete-topic.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet contenant le paramètre `TopicArn` de la rubrique à supprimer pour le transmettre à la méthode `DeleteTopicCommand` de la classe client SNS. Pour appeler la `DeleteTopicCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> *TOPIC_ARN*Remplacez-le par le nom de ressource Amazon (ARN) du sujet que vous supprimez.

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

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Récupération d'attributs de rubrique

Dans cet exemple, utilisez un module Node.js pour récupérer les attributs d'une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `get-topic-attributes.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant le paramètre `TopicArn` d'une rubrique à supprimer pour le transmettre à la méthode `GetTopicAttributesCommand` de la classe client `SNS`. Pour appeler la `GetTopicAttributesCommand` méthode, appelez un objet du service client Amazon SNS et transmettez l'objet de paramètres.

> **Note**
>
> Remplacez *`TOPIC_ARN`* par l'ARN du sujet.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Définition d'attributs de rubrique

Dans cet exemple, utilisez un module Node.js pour définir les attributs modifiables d'une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `set-topic-attributes.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant les paramètres pour la mise à jour de l'attribut, y compris le paramètre `TopicArn` de la rubrique dont vous souhaitez définir les attributs, le nom de l'attribut à définir et la nouvelle valeur pour cet attribut. Vous ne pouvez définir que les attributs `Policy`, `DisplayName` et `DeliveryPolicy`. Transmettez les paramètres à la méthode `SetTopicAttributesCommand` de la classe client SNS. Pour appeler la `SetTopicAttributesCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> Remplacez-le *`ATTRIBUTE_NAME`* par le nom de l'attribut que vous définissez, *`TOPIC_ARN`* par le nom de ressource Amazon (ARN) du sujet dont vous souhaitez définir les attributs et *`NEW_ATTRIBUTE_VALUE`* par la nouvelle valeur de cet attribut.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Publication de messages sur Amazon SNS

Cet exemple de code Node.js présente :

- Comment publier des messages sur une rubrique Amazon SNS.

Scénario

Dans cet exemple, vous utilisez une série de modules Node.js pour publier des messages depuis Amazon SNS vers des points de terminaison, des e-mails ou des numéros de téléphone thématiques. Les modules Node.js utilisent le SDK pour JavaScript envoyer des messages en utilisant cette méthode de la classe SNS client :

- `PublishCommand`

Tâches prérequises

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions indiquées sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment utiliser des objets de service import/export client et des commandes en utilisant ECMAScript6 (ES6).
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> - Si vous préférez utiliser la syntaxe CommonJS, consultezJavaScript ES6Syntaxe / CommonJS.

Publication d'un message dans une rubrique SNS

Dans cet exemple, utilisez un module Node.js pour publier un message sur une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `publish-topic.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant les paramètres de publication d'un message, y compris le texte du message et le nom de ressource Amazon (ARN) d'Amazon SNStopic. Pour plus de détails sur les attributs SMS disponibles, consultez la section Définir SMSAttributes.

Transmettez les paramètres à la `PublishCommand` méthode de la classe `SNS` cliente. Créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> *`MESSAGE_TEXT`*Remplacez-le par le texte du message et *`TOPIC_ARN`* par l'ARN de la rubrique SNS.

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

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *                                                 if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to publish.
 */
export const publish = async (
```

```
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Gestion des abonnements sur Amazon SNS

Cet exemple de code Node.js présente :

- Comment répertorier tous les abonnements à une rubrique Amazon SNS.
- Comment abonner une adresse e-mail, un point de terminaison d'application ou une AWS Lambda fonction à une rubrique Amazon SNS.

- Comment se désabonner des rubriques Amazon SNS.

Scénario

Dans cet exemple, vous utilisez une série de modules Node.js pour publier des messages de notification dans les rubriques Amazon SNS. Les modules Node.js utilisent le SDK pour gérer les sujets JavaScript à l'aide des méthodes suivantes de la classe SNS client :

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

Tâches prérequises

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions indiquées sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment utiliser des objets de service import/export client et des commandes en utilisant ECMAScript6 (ES6).
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> - Si vous préférez utiliser la syntaxe CommonJS, consultezJavaScript ES6Syntaxe / CommonJS.

Liste des abonnements à une rubrique

Dans cet exemple, utilisez un module Node.js pour répertorier tous les abonnements à une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `list-subscriptions-by-topic.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant le paramètre `TopicArn` pour la rubrique dont vous souhaitez répertorier les abonnements. Transmettez les paramètres à la méthode `ListSubscriptionsByTopicCommand` de la classe client SNS. Pour appeler la `ListSubscriptionsByTopicCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS et transmettant l'objet de paramètres.

> **Note**
>
> *`TOPIC_ARN`*Remplacez-le par le Amazon Resource Name (ARN) du sujet dont vous souhaitez répertorier les abonnements.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Abonnement d'une adresse e-mail à une rubrique

Dans cet exemple, utilisez un module Node.js pour abonner une adresse e-mail afin qu'elle reçoive des e-mails SMTP provenant d'une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `subscribe-email.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant le paramètre `Protocol` pour spécifier le protocole `email`, l'élément `TopicArn` pour la rubrique à laquelle s'abonner ainsi qu'une adresse e-mail comme message `Endpoint`. Transmettez les paramètres à la méthode `SubscribeCommand` de la classe client SNS. Vous pouvez utiliser `subscribe` cette méthode pour abonner plusieurs points de terminaison différents à une rubrique Amazon SNS, en fonction des valeurs utilisées pour les paramètres transmis, comme le montreront d'autres exemples présentés dans cette rubrique.

Pour appeler la `SubscribeCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS et transmettant l'objet de paramètres.

> **Note**
>
> *`TOPIC_ARN`*Remplacez-le par le Amazon Resource Name (ARN) du sujet et *`EMAIL_ADDRESS`* par l'adresse e-mail à laquelle vous souhaitez vous abonner.

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

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Confirmation des abonnements

Dans cet exemple, utilisez un module Node.js pour vérifier l'intention du propriétaire d'un terminal de recevoir des e-mails en validant le jeton envoyé au point de terminaison par une action d'abonnement précédente.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

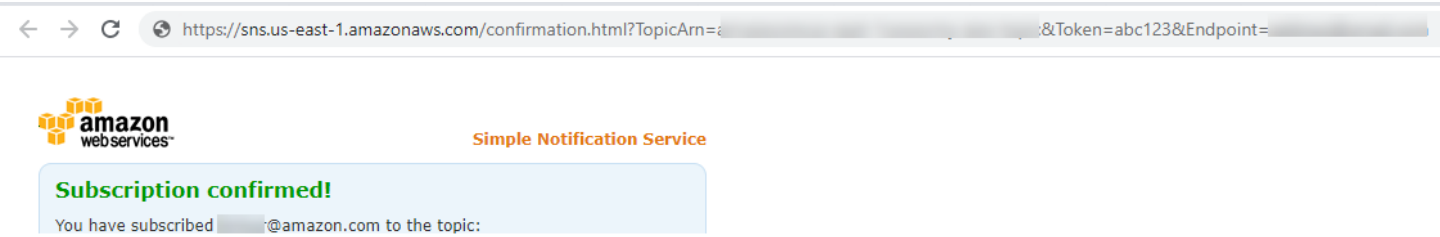
```
import { SNSClient } from "@aws-sdk/client-sns";

// 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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `confirm-subscription.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Définissez les paramètres, y compris le `TOPIC_ARN` et`TOKEN`, et définissez une valeur de `TRUE` ou `FALSE` pour`AuthenticateOnUnsubscribe`.

Le jeton est un jeton de courte durée envoyé au propriétaire d'un point de terminaison lors d'une `SUBSCRIBE` action précédente. Par exemple, pour un point de terminaison de messagerie, `TOKEN` cela se trouve dans l'URL de l'e-mail de confirmation d'abonnement envoyé au propriétaire de l'e-mail. Par exemple, le jeton `abc123` se trouve dans l'URL suivante.



Pour appeler la `ConfirmSubscriptionCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> *`TOPIC_ARN`*Remplacez-le par le Amazon Resource Name (ARN) du sujet, *`TOKEN`* par la valeur du jeton provenant de l'URL envoyée au propriétaire du point de terminaison lors d'une `Subscribe` action précédente, et définissez*`AuthenticateOnUnsubscribe`*. par la valeur `TRUE` ou`FALSE`.

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

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
```

```
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Abonnement d'un point de terminaison d'application à une rubrique

Dans cet exemple, utilisez un module Node.js pour abonner un point de terminaison d'application mobile afin qu'il reçoive des notifications provenant d'une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `subscribe-app.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les modules et packages requis.

Créez un objet contenant le `Protocol` paramètre `TopicArn` pour spécifier le `application` protocole, le sujet auquel vous souhaitez vous abonner et le nom de ressource Amazon (ARN) d'un point de terminaison d'application mobile pour le `Endpoint` paramètre. Transmettez les paramètres à la méthode `SubscribeCommand` de la classe client `SNS`.

Pour appeler la `SubscribeCommand` méthode, créez une fonction asynchrone invoquant un objet de service Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> Remplacez-le *`TOPIC_ARN`* par le Amazon Resource Name (ARN) du sujet et *`MOBILE_ENDPOINT_ARN`* par le point de terminaison auquel vous êtes abonné au sujet.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Abonnement d'une fonction Lambda à une rubrique

Dans cet exemple, utilisez un module Node.js pour abonner une AWS Lambda fonction afin qu'elle reçoive des notifications d'une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `subscribe-lambda.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant le `Protocol` paramètre, en spécifiant le `lambda` protocole, le `TopicArn` sujet auquel vous souhaitez vous abonner et le nom de ressource Amazon (ARN) d'une AWS Lambda fonction en tant que `Endpoint` paramètre. Transmettez les paramètres à la méthode `SubscribeCommand` de la classe client SNS.

Pour appeler la `SubscribeCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> Remplacez-le *`TOPIC_ARN`* par le Amazon Resource Name (ARN) du sujet et *`LAMBDA_FUNCTION_ARN`* par le Amazon Resource Name (ARN) de la fonction Lambda.

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

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Désabonnement d'une rubrique

Dans cet exemple, utilisez un module Node.js pour vous désabonner d'un abonnement à une rubrique Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `unsubscribe.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis.

Créez un objet contenant le `SubscriptionArn` paramètre, en spécifiant le nom de ressource Amazon (ARN) de l'abonnement à désabonner. Transmettez les paramètres à la méthode `UnsubscribeCommand` de la classe client SNS.

Pour appeler la `UnsubscribeCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> *`TOPIC_SUBSCRIPTION_ARN`*Remplacez-le par le Amazon Resource Name (ARN) de l'abonnement pour vous désabonner.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
node unsubscribe.js
```

Cet exemple de code se trouve ici GitHub.

Envoi de SMS avec Amazon SNS

Cet exemple de code Node.js présente :

- Comment obtenir et définir les préférences de messagerie SMS pour Amazon SNS.
- Comment vérifier qu'un numéro de téléphone a désactivé la réception de SMS.
- Comment récupérer une liste de numéros de téléphone ayant désactivé la réception de SMS.
- Comment envoyer un SMS.

Scénario

Vous pouvez utiliser pour envoyer des messages texte, ou des messages SMS, à des appareils compatibles SMS. Vous pouvez envoyer un message directement à un numéro de téléphone, ou vous pouvez envoyer un message à plusieurs numéros de téléphone simultanément en abonnant ces numéros de téléphone à une rubrique et en envoyant votre message à la rubrique.

Dans cet exemple, vous utilisez une série de modules Node.js pour publier des SMS depuis Amazon SNS vers des appareils compatibles SMS. Les modules Node.js utilisent le SDK JavaScript pour publier des messages SMS en utilisant les méthodes suivantes de la classe SNS client :

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

Tâches prérequises

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions indiquées sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment utiliser des objets de service import/export client et des commandes en utilisant ECMAScript6 (ES6).
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> - Si vous préférez utiliser la syntaxe CommonJS, consultezJavaScript ES6Syntaxe / CommonJS.

Récupération d'attributs SMS

Utilisez Amazon SNS pour définir vos préférences en matière de messagerie SMS, telles que la manière dont vos envois sont optimisés (en termes de coût ou de fiabilité), votre limite de dépenses mensuelles, la manière dont les envois de messages sont enregistrés et si vous souhaitez vous abonner aux rapports quotidiens d'utilisation des SMS. Ces préférences sont récupérées et définies sous forme d'attributs SMS pour Amazon SNS.

Dans cet exemple, utilisez un module Node.js pour obtenir les attributs SMS actuels dans Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `get-sms-attributes.js`.

Configurez le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Créez un objet contenant les paramètres pour récupérer les attributs SMS, y compris les noms des attributs individuels. Pour plus de détails sur les attributs SMS disponibles, consultez la section Définir SMSAttributes dans le manuel Amazon Simple Notification Service API Reference.

Cet exemple récupère l'attribut `DefaultSMSType`, qui contrôle si les messages SMS sont envoyés en tant que `Promotional`, ce qui optimise la transmission des messages au plus bas coût ou en tant que `Transactional`, ce qui optimise la transmission des messages à une fiabilité optimale. Transmettez les paramètres à la méthode `SetTopicAttributesCommand` de la classe client `SNS`. Pour appeler la `SetSMSAttributesCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> Remplacez *`ATTRIBUTE_NAME`* par le nom de l'attribut.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Définition d'attributs SMS

Dans cet exemple, utilisez un module Node.js pour obtenir les attributs SMS actuels dans Amazon SNS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `set-sms-attribute-type.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez un objet contenant les paramètres pour définir les attributs SMS, y compris les noms des attributs individuels et les valeurs de chacun d'entre eux. Pour plus de détails sur les attributs SMS disponibles, consultez la section Définir SMSAttributes dans le manuel Amazon Simple Notification Service API Reference.

Cet exemple définit l'attribut `DefaultSMSType` sur `Transactional`, ce qui optimise la transmission de message à une fiabilité optimale. Transmettez les paramètres à la méthode `SetTopicAttributesCommand` de la classe client `SNS`. Pour appeler la `SetSMSAttributesCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Vérification d'un numéro de téléphone désactivé

Dans cet exemple, utilisez un module Node.js pour vérifier qu'un numéro de téléphone a désactivé la réception de SMS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *REGION* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `check-if-phone-number-is-opted-out.js`. Configurez le kit SDK comme illustré précédemment. Créez un objet contenant le numéro de téléphone à vérifier en tant que paramètre.

Cet exemple définit le paramètre `PhoneNumber` pour spécifier le numéro de téléphone à vérifier. Transmettez l'objet à la méthode `CheckIfPhoneNumberIsOptedOutCommand` de la classe client SNS. Pour appeler la `CheckIfPhoneNumberIsOptedOutCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> 1.
>
> Remplacez *PHONE_NUMBER* par le numéro de téléphone.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Liste des numéros de téléphone désactivés

Dans cet exemple, utilisez un module Node.js pour récupérer une liste des numéros de téléphone ayant désactivé la réception de SMS.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *`REGION`* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `list-phone-numbers-opted-out.js`. Configurez le kit SDK comme illustré précédemment. Créez un objet vide comme paramètre.

Transmettez l'objet à la méthode `ListPhoneNumbersOptedOutCommand` de la classe client `SNS`. Pour appeler la `ListPhoneNumbersOptedOutCommand` méthode, créez une fonction asynchrone invoquant un objet de service client Amazon SNS, en transmettant l'objet de paramètres.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Publication d'un SMS

Dans cet exemple, utilisez un module Node.js pour envoyer un SMS à un numéro de téléphone.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`snsClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon SNS. Remplacez *REGION* par votre AWS région.

```
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({});
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `publish-sms.js`. Configurez le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez un objet contenant les paramètres `Message` et `PhoneNumber`.

Lorsque vous envoyez un SMS, spécifiez le numéro de téléphone au format E.164. E.164 est une norme pour la structure des numéros de téléphone, qui est utilisée pour les télécommunications internationales. Les numéros qui respectent ce format peuvent comporter 15 chiffres au maximum et commencent par le caractère plus (+) et le code pays. Par exemple, un numéro de téléphone américain au format E.164 s'affichera sous la forme +1001 0100XXX555.

Cet exemple définit le paramètre `PhoneNumber` pour spécifier le numéro de téléphone qui envoie le message. Transmettez l'objet à la méthode `PublishCommand` de la classe client `SNS`. Pour appeler la `PublishCommand` méthode, créez une fonction asynchrone invoquant un objet de service Amazon SNS, en transmettant l'objet de paramètres.

> **Note**
>
> Remplacez *TEXT_MESSAGE* par le message texte et *PHONE_NUMBER* par le numéro de téléphone.

```
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;
};
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici GitHub.

Exemples d'Amazon Transcribe

Amazon Transcribe permet aux développeurs d'ajouter facilement des fonctionnalités de synthèse vocale à leurs applications.

L' JavaScript API d'Amazon Transcribe est exposée via la classe TranscribeServiceclient.

Rubriques

- Exemples d'Amazon Transcribe
- Exemples médicaux d'Amazon Transcribe

Exemples d'Amazon Transcribe

Dans cet exemple, une série de modules Node.js sont utilisés pour créer, répertorier et supprimer des tâches de transcription à l'aide des méthodes suivantes de la classe `TranscribeService` client :

- `StartTranscriptionJobCommand`
- `ListTranscriptionJobsCommand`
- `DeleteTranscriptionJobCommand`

Pour plus d'informations sur les utilisateurs d'Amazon Transcribe, consultez le guide du développeur Amazon Transcribe.

Tâches prérequises

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions figurant sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez

la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets de service client et des commandes à l'aide de ECMAScript6 ()ES6.
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> - Si vous préférez utiliser la syntaxe CommonJS, voir JavaScript ES6Syntaxe /CommonJS

Démarrage d'une tâche sur Amazon Transcribe

Cet exemple montre comment démarrer une tâche de transcription Amazon Transcribe à l'aide du. AWS SDK pour JavaScript Pour de plus amples informations, veuillez consulter StartTranscriptionJobCommand.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`transcribeClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Transcribe. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `transcribe-create-job.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez un objet de paramètres en spécifiant les paramètres requis. Démarrez la tâche à l'aide de la `StartMedicalTranscriptionJobCommand` commande.

> **Note**
>
> *MEDICAL_JOB_NAME*Remplacez-le par le nom de la tâche de transcription. Pour *OUTPUT_BUCKET_NAME* spécifier le compartiment Amazon S3 dans lequel la sortie est enregistrée. Pour *JOB_TYPE* spécifier les types de tâches. Pour *SOURCE_LOCATION* spécifier l'emplacement du fichier source. Pour *SOURCE_FILE_LOCATION* spécifier l'emplacement du fichier multimédia d'entrée.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Répertorier les offres d'emploi d'Amazon Transcribe

Cet exemple montre comment répertorier les tâches de transcription Amazon Transcribe à l'aide du. AWS SDK pour JavaScript Pour plus d'informations sur les autres paramètres que vous pouvez modifier, consultez `ListTranscriptionJobCommand`.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`transcribeClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Transcribe. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `transcribe-list-jobs.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez un objet de paramètres avec les paramètres requis.

> **Note**
>
> *`KEY_WORD`*Remplacez-le par un mot clé que le nom des tâches renvoyées doit contenir.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Supprimer une tâche Amazon Transcribe

Cet exemple montre comment supprimer une tâche de transcription Amazon Transcribe à l'aide du. AWS SDK pour JavaScript Pour plus d'informations sur les options facultatives, consultez `DeleteTranscriptionJobCommand`.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`transcribeClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Transcribe. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `transcribe-delete-job.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Spécifiez la AWS région et le nom de la tâche que vous souhaitez supprimer.

> **Note**
>
> Remplacez *JOB_NAME* par le nom de la tâche à supprimer.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Exemples médicaux d'Amazon Transcribe

Dans cet exemple, une série de modules Node.js sont utilisés pour créer, répertorier et supprimer des tâches de transcription médicale à l'aide des méthodes suivantes de la classe `TranscribeService` client :

- `StartMedicalTranscriptionJobCommand`
- `ListMedicalTranscriptionJobsCommand`
- `DeleteMedicalTranscriptionJobCommand`

Pour plus d'informations sur les utilisateurs d'Amazon Transcribe, consultez le guide du développeur Amazon Transcribe.

Tâches prérequises

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions figurant sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets de service client et des commandes à l'aide de ECMAScript6 ()ES6.
>
> - Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> - Si vous préférez utiliser la syntaxe CommonJS, voir JavaScript ES6Syntaxe /CommonJS

Démarrage d'une tâche de transcription médicale sur Amazon Transcribe

Cet exemple montre comment démarrer une tâche de transcription médicale Amazon Transcribe à l'aide du. AWS SDK pour JavaScript Pour plus d'informations, consultez startMedicalTranscriptionJob.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`transcribeClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Transcribe. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `transcribe-create-medical-job.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez un objet de paramètres en spécifiant les paramètres requis. Commencez le travail médical à l'aide de la `StartMedicalTranscriptionJobCommand` commande.

> **Note**
>
> *`MEDICAL_JOB_NAME`*Remplacez-le par un nom pour le travail de transcription médicale. Pour *`OUTPUT_BUCKET_NAME`* spécifier le compartiment Amazon S3 dans lequel la sortie est enregistrée. Pour *`JOB_TYPE`* spécifier les types de tâches. Pour *`SOURCE_LOCATION`* spécifier l'emplacement du fichier source. Pour *`SOURCE_FILE_LOCATION`* spécifier l'emplacement du fichier multimédia d'entrée.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Liste des offres d'emploi dans le secteur médical d'Amazon Transcribe

Cet exemple montre comment répertorier les tâches de transcription Amazon Transcribe à l'aide du. AWS SDK pour JavaScript Pour de plus amples informations, veuillez consulter ListTranscriptionMedicalJobsCommand.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`transcribeClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Transcribe. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `transcribe-list-medical-jobs.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez un objet de paramètres avec les paramètres requis et listez les tâches médicales à l'aide de la `ListMedicalTranscriptionJobsCommand` commande.

> **Note**
>
> *KEYWORD*Remplacez-le par un mot clé que le nom des tâches renvoyées doit contenir.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Supprimer un emploi médical sur Amazon Transcribe

Cet exemple montre comment supprimer une tâche de transcription Amazon Transcribe à l'aide du. AWS SDK pour JavaScript Pour plus d'informations sur les options facultatives, consultez `DeleteTranscriptionMedicalJobCommand`.

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`transcribeClient.js`. Copiez-collez le code ci-dessous pour créer l'objet client Amazon Transcribe. Remplacez *`REGION`* par votre AWS région.

```
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 };
```

Cet exemple de code se trouve ici GitHub.

Créez un module Node.js nommé `transcribe-delete-job.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Créez un objet de paramètres avec les paramètres requis et supprimez le travail médical à l'aide de la `DeleteMedicalJobCommand` commande.

> **Note**
>
> Remplacez *`JOB_NAME`* par le nom de la tâche à supprimer.

```
// 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();
```

Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

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

Cet exemple de code se trouve ici sur GitHub.

Configuration de Node.js sur une EC2 instance Amazon

Un scénario courant d'utilisation de Node.js avec le SDK pour JavaScript consiste à configurer et à exécuter une application Web Node.js sur une instance Amazon Elastic Compute Cloud (Amazon EC2). Dans ce didacticiel, vous allez créer une instance Linux, vous y connecter à l'aide de SSH et installer Node.js pour que ce dernier s'exécute dans cette instance.

Conditions préalables

Ce didacticiel part du principe que vous avez déjà lancé une instance Linux avec un nom DNS public accessible depuis Internet et à laquelle vous pouvez vous connecter via SSH. Pour plus d'informations, consultez Étape 1 : Lancer une instance dans le guide de EC2 l'utilisateur Amazon.

> **Important**
>
> Utilisez l'Amazon Machine Image (AMI) Amazon Linux 2023 lors du lancement d'une nouvelle EC2 instance Amazon.

Vous devez aussi avoir configuré votre groupe de sécurité pour permettre les connexions SSH (port 22), HTTP (port 80) et HTTPS (port 443). Pour plus d'informations sur ces prérequis, consultez la section Configuration avec Amazon EC2 dans le guide de l' EC2 utilisateur Amazon.

Procédure

La procédure suivante vous aide à installer Node.js sur une instance Amazon Linux. Vous pouvez utiliser ce serveur pour héberger une application web Node.js.

Pour configurez Node.js sur votre instance Linux

1. Connectez-vous à votre instance Linux en tant que `ec2-user` à l'aide de SSH.
2. Installez le gestionnaire de version de nœud (nvm) en saisissant ce qui suit sur la ligne de commande.

   > **Warning**
   >
   > AWS ne contrôle pas le code suivant. Avant de l'exécuter, vérifiez son authenticité et son intégrité. Vous trouverez plus d'informations sur ce code dans le GitHub dépôt nvm.

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

   Nous allons l'utiliser nvm pour installer Node.js car il nvm peut installer plusieurs versions de Node.js et vous permettre de passer de l'une à l'autre.
3. Chargez nvm en saisissant ce qui suit sur la ligne de commande.

   ```
   source ~/.bashrc
   ```
4. Utilisez nvm pour installer la dernière version LTS de Node.js en tapant ce qui suit sur la ligne de commande.

```
nvm install --lts
```

L'installation de Node.js installe également le Node Package Manager (npm) afin que vous puissiez installer des modules supplémentaires selon vos besoins.

5. Testez l'installation et le fonctionnement de Node.js en saisissant ce qui suit dans la ligne de commande.

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

Le message suivant affiche alors la version de Node.js qui est en cours d'exécution.

```
Running Node.js VERSION
```

> **Note**
>
> L'installation du nœud ne s'applique qu'à la EC2 session Amazon en cours. Si vous redémarrez votre session CLI, vous devez réutiliser nvm pour activer la version du nœud installé. Si l'instance est résiliée, vous devez réinstaller le nœud. L'alternative est de créer une Amazon Machine Image (AMI) de l' EC2 instance Amazon une fois que vous avez obtenu la configuration que vous souhaitez conserver, comme décrit dans la rubrique suivante.

Création d'une image machine Amazon (AMI)

Après avoir installé Node.js sur une EC2 instance Amazon, vous pouvez créer une Amazon Machine Image (AMI) à partir de cette instance. La création d'une AMI facilite le provisionnement de plusieurs EC2 instances Amazon avec la même installation Node.js. Pour plus d'informations sur la création d'une AMI à partir d'une instance existante, consultez la section Création d'une AMI Linux basée sur Amazon EBS dans le guide de EC2 l'utilisateur Amazon.

Ressources connexes

Pour plus d'informations sur les commandes et les logiciels utilisés dans cette rubrique, consultez les pages Web suivantes :

- Gestionnaire de versions de nœuds (nvm) —Voir nvm repo on. GitHub

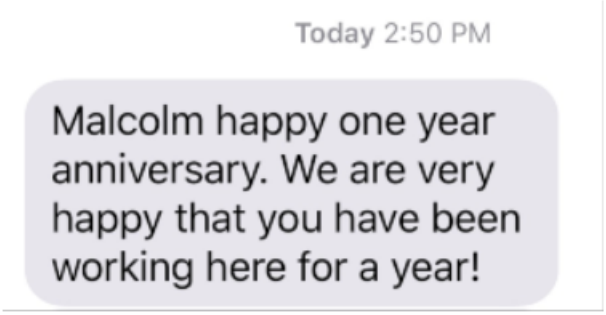
- Node Package Manager (npm) —Voir le site Web de npm.

Invoquer Lambda avec API Gateway

Vous pouvez appeler une fonction Lambda à l'aide d'Amazon API Gateway, qui est un AWS service de création, de publication, de maintenance, de surveillance et de sécurisation des protocoles REST, HTTP et WebSocket APIs à grande échelle. Les développeurs d'API peuvent créer APIs cet accès AWS ou d'autres services Web, ainsi que des données stockées dans le AWS cloud. En tant que développeur d'API Gateway, vous pouvez créer des applications APIs pour les utiliser dans vos propres applications clientes. Pour plus d'informations, consultez Qu'est-ce qu'Amazon API Gateway ?

AWS Lambda est un service de calcul qui vous permet d'exécuter du code sans provisionner ni gérer de serveurs. Vous pouvez créer des fonctions Lambda dans différents langages de programmation. Pour plus d'informations AWS Lambda, voir Qu'est-ce que AWS Lambda.

Dans cet exemple, vous créez une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Supposons, par exemple, qu'une organisation envoie un message texte mobile à ses employés pour les féliciter à la date du premier anniversaire, comme le montre cette illustration.



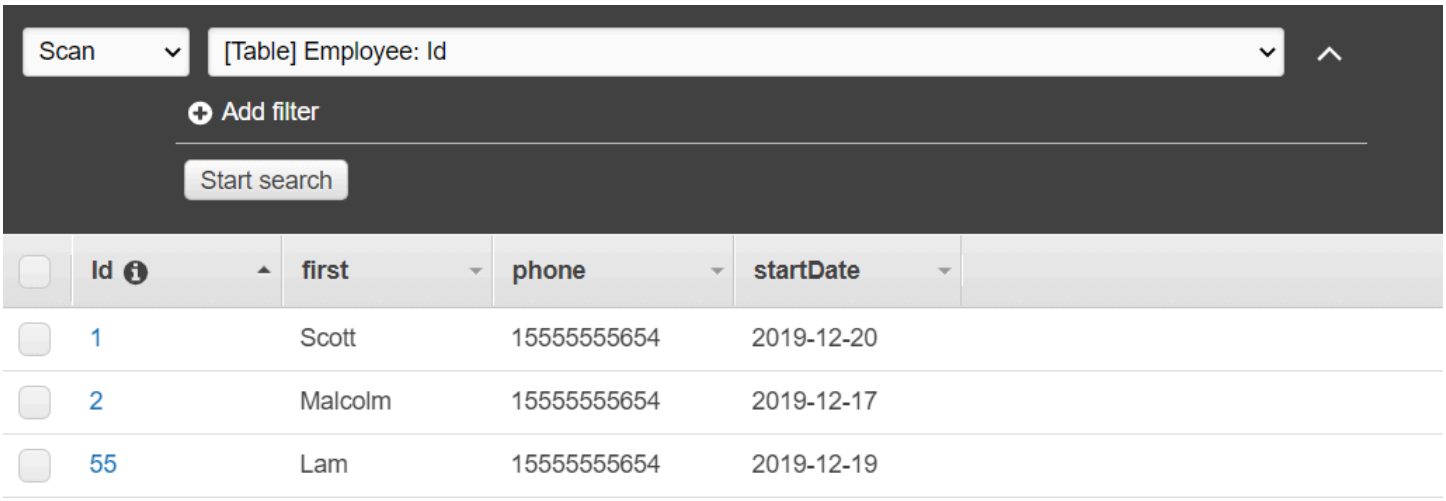
La réalisation de l'exemple devrait prendre environ 20 minutes.

Cet exemple montre comment utiliser la JavaScript logique pour créer une solution qui exécute ce cas d'utilisation. Par exemple, vous apprendrez à lire une base de données pour déterminer quels employés ont atteint le premier anniversaire, à traiter les données et à envoyer un message texte à l'aide d'une fonction Lambda. Vous apprendrez ensuite à utiliser API Gateway pour appeler cette AWS Lambda fonction à l'aide d'un point de terminaison Rest. Par exemple, vous pouvez appeler la fonction Lambda à l'aide de cette commande curl :

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/
employee"
```

Ce AWS didacticiel utilise une table Amazon DynamoDB nommée Employee qui contient ces champs.

- id - la clé primaire de la table.
- FirstName : prénom de l'employé.
- téléphone - numéro de téléphone de l'employé.
- Date de début : date de début de l'employé.



> **Important**
>
> Coût de réalisation : Les AWS services inclus dans ce document sont inclus dans le niveau AWS gratuit. Veillez toutefois à désactiver toutes les ressources une fois que vous aurez terminé cet exemple afin de ne pas être débité.

Pour créer l'application, procédez comme suit :

1. Compléter les prérequis
2. Créez les AWS ressources
3. Préparez le script du navigateur
4. Création et téléchargement de la fonction Lambda

5. Déployer la fonction Lambda
6. Exécutez l'application
7. Supprimer les ressources

Tâches préalables

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions figurant sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

Créez les AWS ressources

Ce didacticiel nécessite les ressources suivantes :

- Une table Amazon DynamoDB `Employee` nommée avec une clé `Id` nommée et les champs illustrés dans l'illustration précédente. Assurez-vous de saisir les données correctes, y compris un téléphone portable valide avec lequel vous souhaitez tester ce cas d'utilisation. Pour plus d'informations, consultez la section Création d'une table.
- Rôle IAM associé à des autorisations permettant d'exécuter des fonctions Lambda.
- Un compartiment Amazon S3 pour héberger la fonction Lambda.

Vous pouvez créer ces ressources manuellement, mais nous vous recommandons de les approvisionner à l'aide de la méthode CloudFormation décrite dans ce didacticiel.

Créez les AWS ressources à l'aide de CloudFormation

CloudFormation vous permet de créer et de provisionner des déploiements AWS d'infrastructure de manière prévisible et répétée. Pour plus d'informations CloudFormation, consultez le guide de AWS CloudFormation l'utilisateur.

Pour créer la CloudFormation pile à l'aide de AWS CLI :

1. Installez et configurez en AWS CLI suivant les instructions du guide de l'AWS CLI utilisateur.
2. Créez un fichier nommé `setup.yaml` dans le répertoire racine du dossier de votre projet et GitHubcopiez-y le contenu.

   > **Note**
   >
   > Le CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour plus d'informations à ce sujet AWS CDK, consultez le guide du AWS Cloud Development Kit (AWS CDK) développeur.

3. Exécutez la commande suivante depuis la ligne de commande, en la *`STACK_NAME`* remplaçant par un nom unique pour la pile.

   > **Important**
   >
   > Le nom de la pile doit être unique au sein d'une AWS région et d'un AWS compte. Vous pouvez spécifier jusqu'à 128 caractères. Les chiffres et les tirets sont autorisés.

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

   Pour plus d'informations sur les paramètres de `create-stack` commande, consultez le guide de référence des AWS CLI commandes et le guide de CloudFormation l'utilisateur.
4. Ensuite, renseignez le tableau en suivant la procédureRemplissage du tableau.

Remplissage du tableau

Pour remplir le tableau, créez d'abord un répertoire nommé`libs`, puis créez un fichier nommé`dynamoClient.js`, puis collez-y le contenu ci-dessous.

```
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 };
```

Ce code est disponible ici GitHub.

Créez ensuite un fichier nommé `populate-table.js` dans le répertoire racine du dossier de votre projet et GitHubcopiez-y le contenu. Pour l'un des articles, remplacez la valeur de la `phone` propriété par un numéro de téléphone portable valide au format E.164, et la valeur `startDate` par la date du jour.

Exécutez la commande suivante depuis la ligne de commande.

```
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();
```

Ce code est disponible ici GitHub.

Création de la AWS Lambda fonction

Configuration du kit SDK

Dans le `libs` répertoire, créez des fichiers nommés `snsClient.js` et`lambdaClient.js`, puis collez le contenu ci-dessous dans ces fichiers, respectivement.

```
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 };
```

Remplacez *`REGION`* par la AWS région. Ce code est disponible ici GitHub.

```
const { LambdaClient } = require("@aws-sdk/client-lambda");
```

```
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Lambda service client object.
const lambdaClient = new LambdaClient({ region: REGION });
module.exports = { lambdaClient };
```

Remplacez *REGION* par la AWS région. Ce code est disponible ici GitHub.

Importez d'abord les modules et commandes requis AWS SDK pour JavaScript (v3). Calculez ensuite la date du jour et attribuez-la à un paramètre. Troisièmement, créez les paramètres pour`ScanCommand`. *TABLE_NAME*Remplacez-le par le nom de la table que vous avez créée dans la Créez les AWS ressources section de cet exemple.

L'extrait de code suivant illustre cette étape. (Pour obtenir l'exemple complet, consultez Regroupement de la fonction Lambda.)

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

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

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

Analyse de la table DynamoDB

Tout d'abord, créez une async/await fonction appelée `sendText` pour publier un message texte à l'aide d'Amazon SNS`PublishCommand`. Ajoutez ensuite un schéma de `try` blocs qui analyse la table DynamoDB à la recherche des employés à l'occasion de leur anniversaire de travail aujourd'hui, puis appelle `sendText` la fonction pour envoyer un message texte à ces employés. En cas d'erreur, le `catch` bloc est appelé.

L'extrait de code suivant illustre cette étape. (Pour obtenir l'exemple complet, consultez Regroupement de la fonction Lambda.)

```
// Helper function to send message using Amazon SNS.
exports.handler = async () => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      await snsClient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to identify employees with work anniversary today.
    const data = await dynamoClient.send(new ScanCommand(params));
    for (const element of data.Items) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message: `Hi ${element.firstName.S}; congratulations on your work anniversary!
`,
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    }
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Regroupement de la fonction Lambda

Cette rubrique décrit comment regrouper `mylambdafunction.ts` les AWS SDK pour JavaScript modules requis pour cet exemple dans un fichier groupé appelé`index.js`.

1. Si ce n'est pas déjà fait, suivez cet exemple Tâches préalables pour installer le webpack.

   > **Note**
   >
   > Pour plus d'informations sur Webpack, consultezRegroupez des applications avec Webpack.

2. Exécutez ce qui suit dans la ligne de commande JavaScript pour regrouper les informations de cet exemple dans un fichier appelé <`index.js`> :

   ```
   webpack mylambdafunction.ts --mode development --target node --devtool false --
   output-library-target umd -o index.js
   ```

   > **Important**
   >
   > Notez que la sortie est nomméeindex.js. Cela est dû au fait que les fonctions Lambda doivent avoir un `index.js` gestionnaire pour fonctionner.

3. Compressez le fichier de sortie groupé`index.js`,, dans un fichier ZIP nommé`mylambdafunction.zip`.
4. `mylambdafunction.zip`Téléchargez-le dans le compartiment Amazon S3 que vous avez créé dans la Créez les AWS ressources rubrique de ce didacticiel.

Déployez la fonction Lambda.

À la racine de votre projet, créez un `lambda-function-setup.ts` fichier et collez-y le contenu ci-dessous.

*BUCKET_NAME*Remplacez-le par le nom du compartiment Amazon S3 dans lequel vous avez chargé la version ZIP de votre fonction Lambda. Remplacez *ZIP_FILE_NAME* par le nom la version ZIP de votre fonction Lambda. *ROLE*Remplacez-le par le numéro de ressource Amazon (ARN) du rôle IAM que vous avez créé dans le Créez les AWS ressources sujet de ce didacticiel. Remplacez *LAMBDA_FUNCTION_NAME* par le nom de la fonction Lambda.

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand
} = require ( "@aws-sdk/client-lambda" );
const { lambdaClient} = require ( "./libs/lambdaClient.js );

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
 Services (Amazon SNS) to " +
    "send employees an email on each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambdaClient.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

Entrez ce qui suit sur la ligne de commande pour déployer la fonction Lambda.

```
node lambda-function-setup.ts
```

Cet exemple de code est disponible ici GitHub.

Configurer API Gateway pour appeler la fonction Lambda

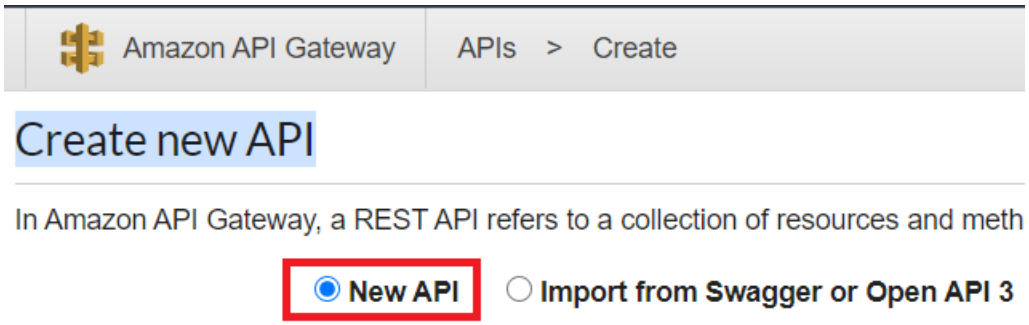
Pour créer l'application, procédez comme suit :

1. Création de l'API Rest
2. Testez la méthode API Gateway
3. Déployer la méthode API Gateway

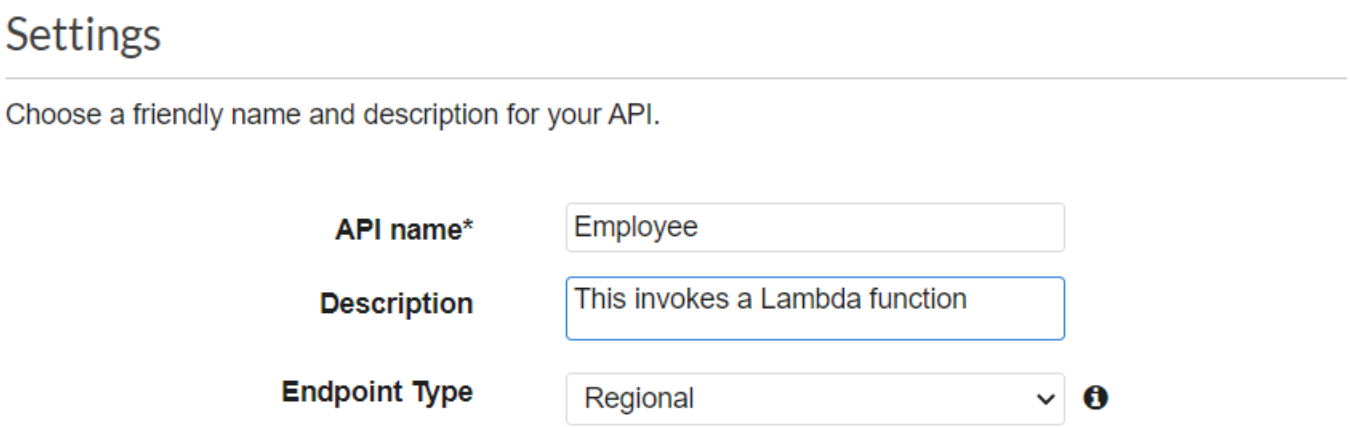
Création de l'API Rest

Vous pouvez utiliser la console API Gateway pour créer un point de terminaison REST pour la fonction Lambda. Une fois cela fait, vous pouvez invoquer la fonction Lambda à l'aide d'un appel RESTful.

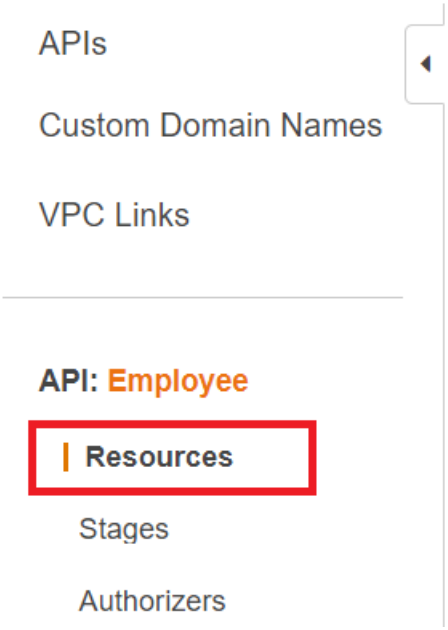
1. Connectez-vous à la console Amazon API Gateway.
2. Sous Rest API, choisissez Build.
3. Sélectionnez Nouvelle API.



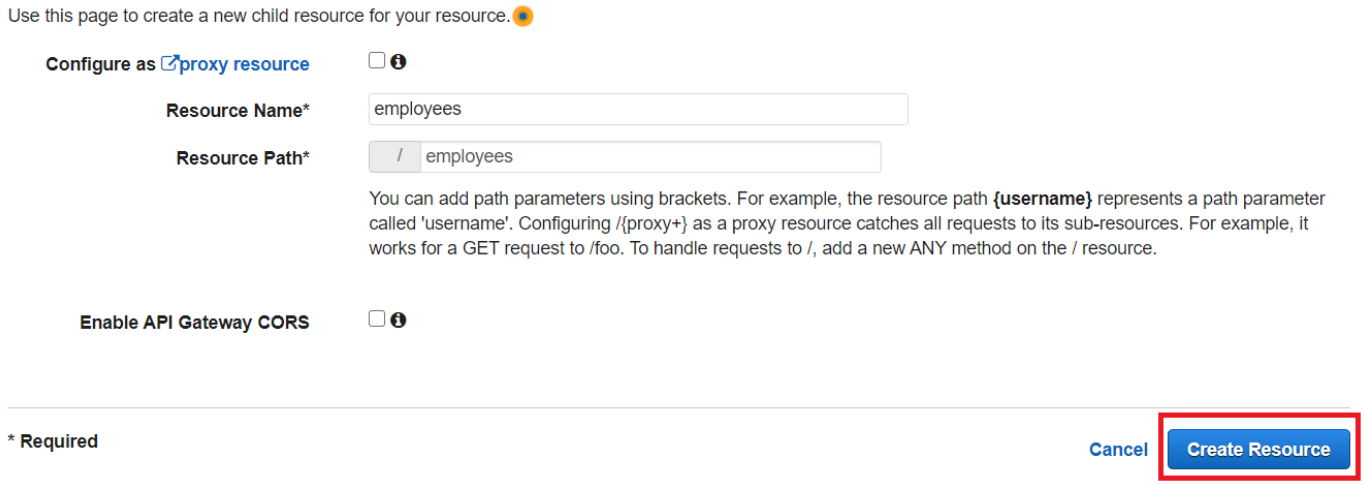
4. Spécifiez Employee comme nom d'API et fournissez une description.



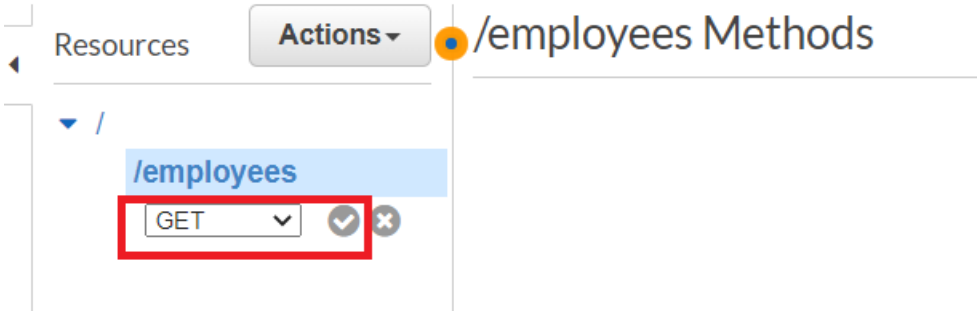
5. Sélectionnez Create API (Créer une API).
6. Choisissez Ressources dans la section Employé.



7. Dans le champ du nom, spécifiez les employés.
8. Choisissez Create Resources (Créer des ressources).
9. Dans le menu déroulant Actions, choisissez Create Resources.



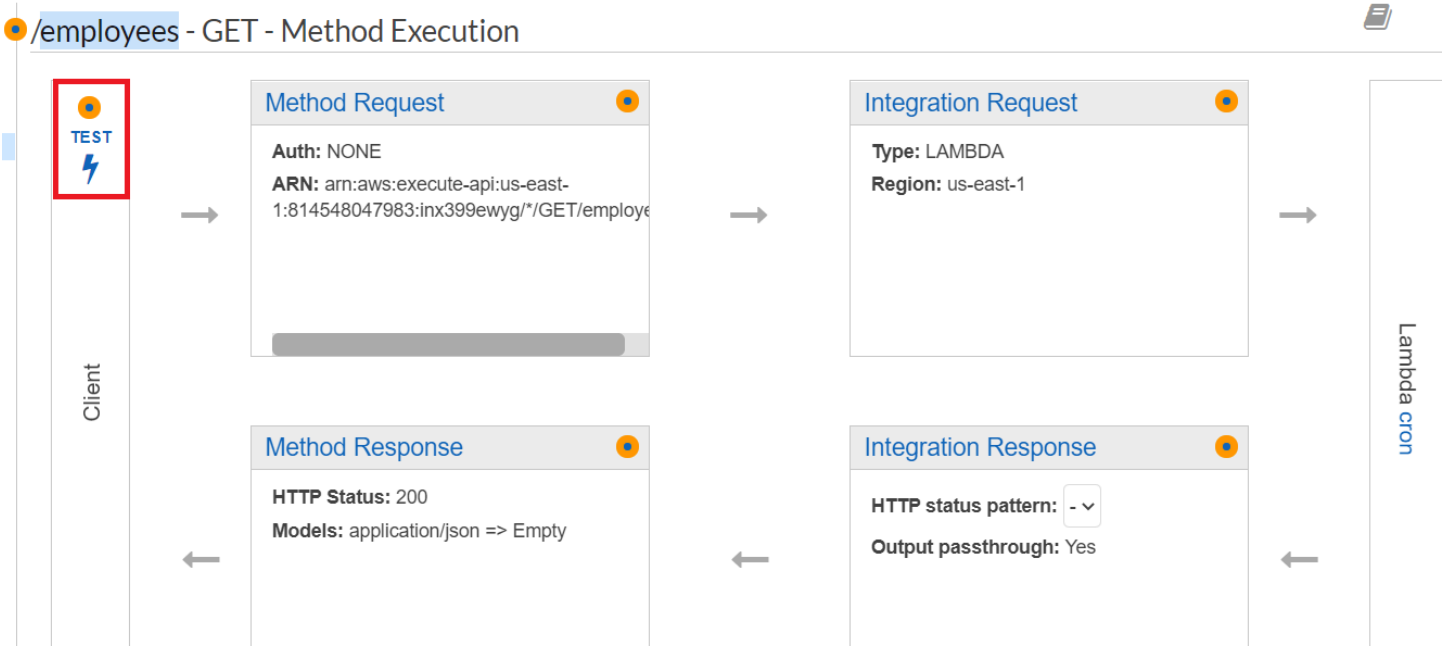
10. Choisissez /employees, sélectionnez Create Method dans les Actions, puis sélectionnez GET dans le menu déroulant situé sous /employees. Choisissez l'icône représentant une coche.



11. Choisissez fonction Lambda et entrez mylambdafunction comme nom de la fonction Lambda. Choisissez Enregistrer.

Testez la méthode API Gateway

À ce stade du didacticiel, vous pouvez tester la méthode API Gateway qui appelle la fonction Lambda mylambdafunction. Pour tester la méthode, choisissez Test, comme indiqué dans l'illustration suivante.

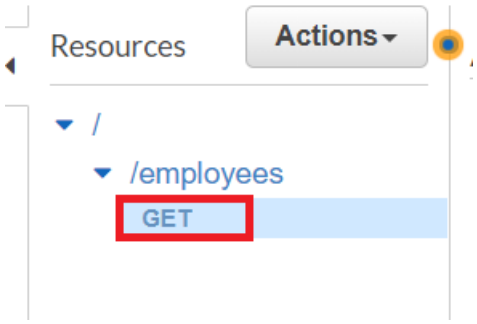


Une fois que la fonction Lambda est invoquée, vous pouvez consulter le fichier journal pour voir s'il s'agit d'un message de réussite.

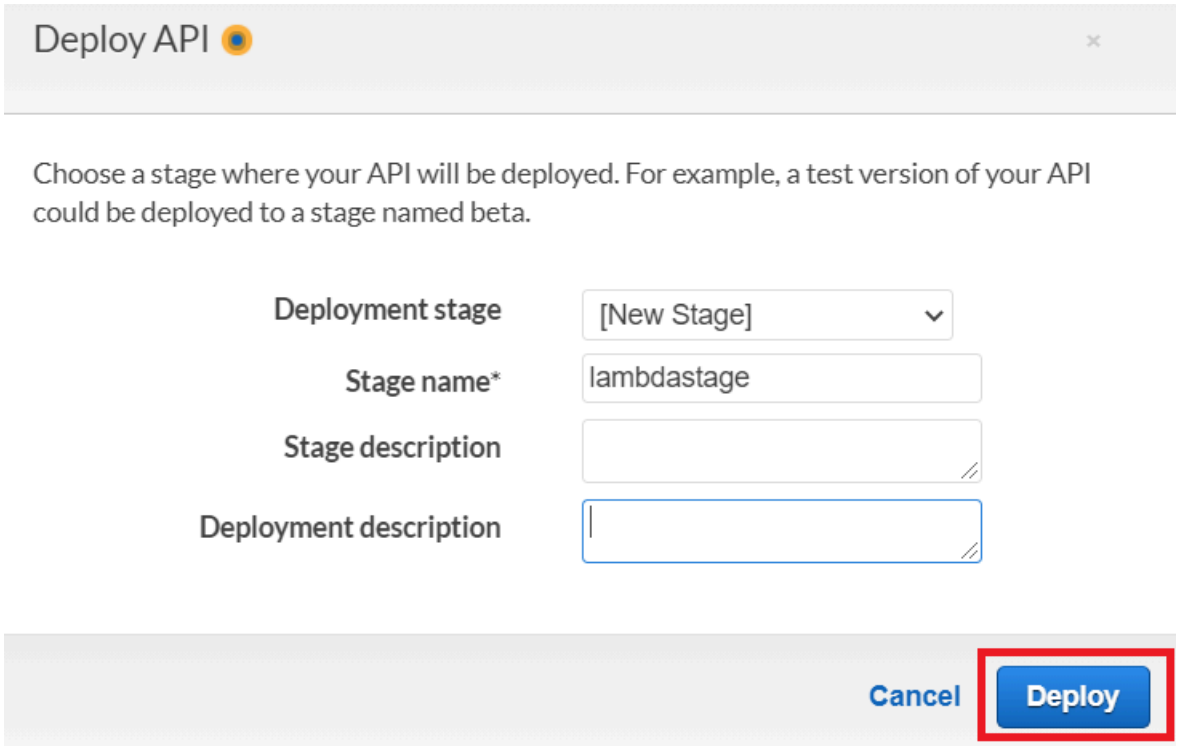
Déployer la méthode API Gateway

Une fois le test réussi, vous pouvez déployer la méthode depuis la console Amazon API Gateway.

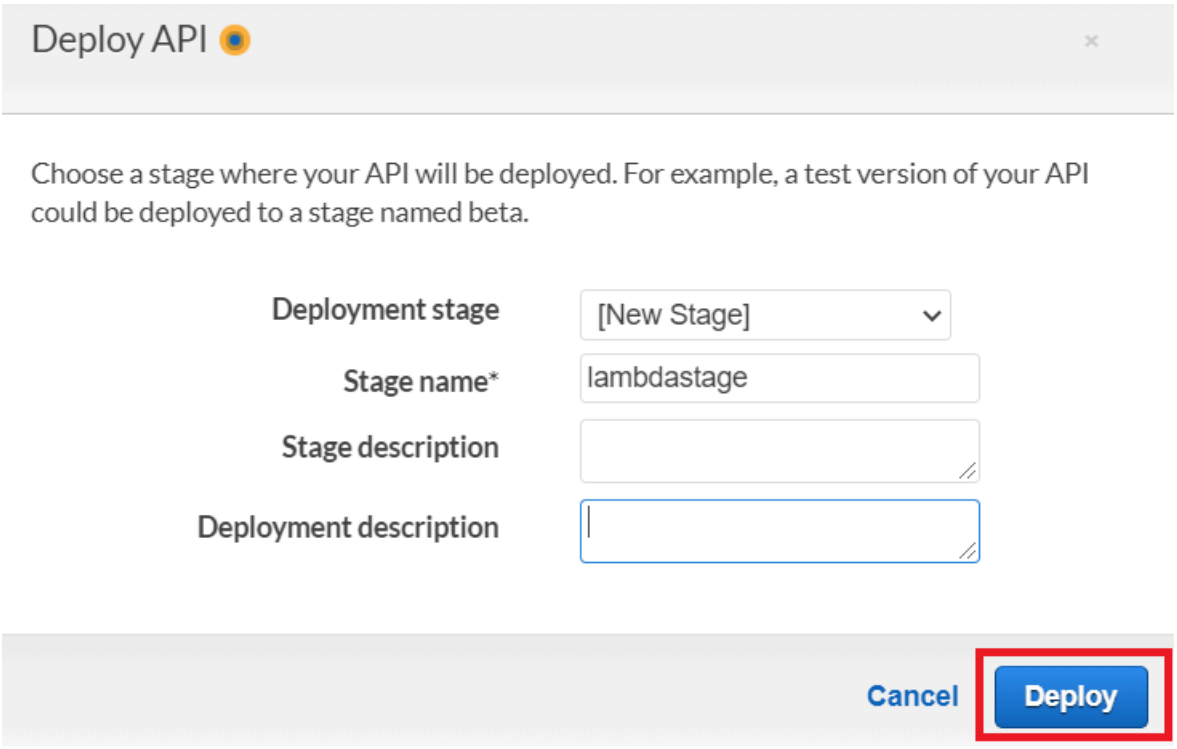
1. Choisissez Get.



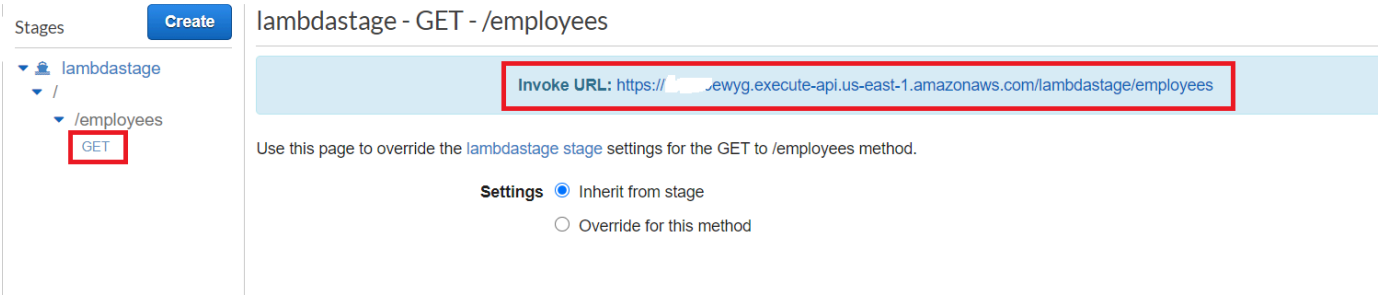
2. Dans le menu déroulant Actions, sélectionnez Déployer l'API.



3. Remplissez le formulaire Deploy API et choisissez Deploy.



4. Choisissez Save Changes (Enregistrer les modifications).
5. Choisissez à nouveau Obtenir et remarquez que l'URL change. Il s'agit de l'URL d'appel que vous pouvez utiliser pour appeler la fonction Lambda.



Supprimer les ressources

Félicitations ! Vous avez invoqué une fonction Lambda via Amazon API Gateway à l'aide du. AWS SDK pour JavaScript Comme indiqué au début de ce didacticiel, veillez à désactiver toutes les ressources que vous créez pendant que vous suivez ce didacticiel pour vous assurer que vous n'êtes pas débité. Pour ce faire, supprimez la CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

1. Ouvrez le CloudFormation dans la console AWS de gestion.

2. Ouvrez la page Stacks, puis sélectionnez la pile.
3. Sélectionnez Supprimer.

Création d'événements planifiés pour exécuter AWS Lambda des fonctions

Vous pouvez créer un événement planifié qui invoque une AWS Lambda fonction à l'aide d'un CloudWatch événement Amazon. Vous pouvez configurer un CloudWatch événement pour utiliser une expression cron afin de planifier le moment où une fonction Lambda est invoquée. Par exemple, vous pouvez planifier un CloudWatch événement pour appeler une fonction Lambda tous les jours de la semaine.

AWS Lambda est un service de calcul qui vous permet d'exécuter du code sans provisionner ni gérer de serveurs. Vous pouvez créer des fonctions Lambda dans différents langages de programmation. Pour plus d'informations AWS Lambda, voir Qu'est-ce que AWS Lambda.

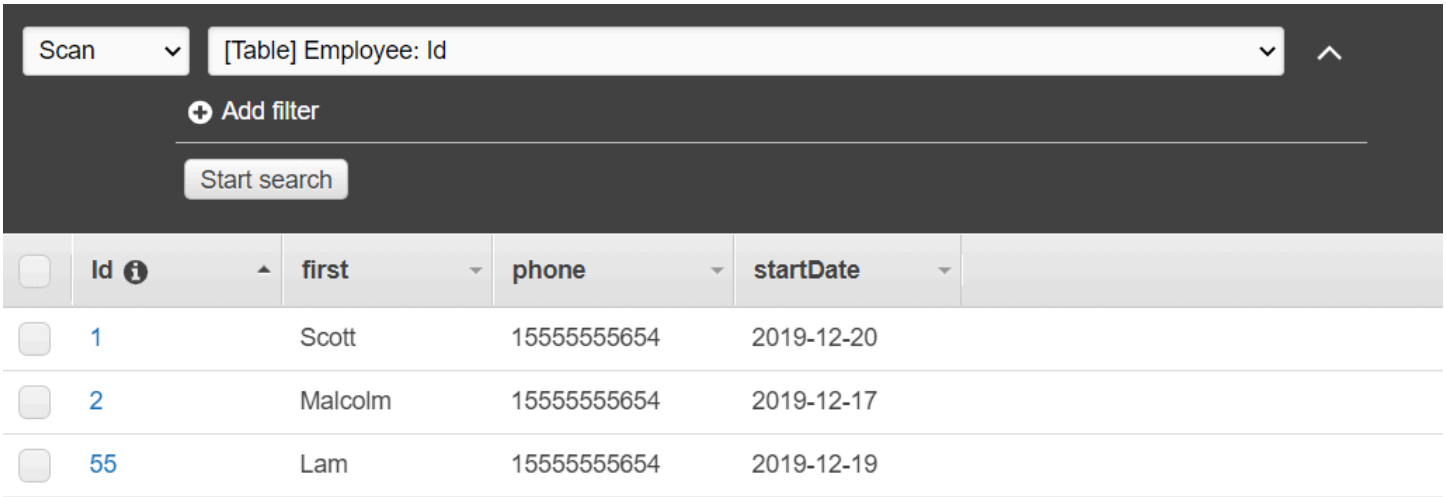
Dans ce didacticiel, vous allez créer une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Supposons, par exemple, qu'une organisation envoie un message texte mobile à ses employés pour les féliciter à la date du premier anniversaire, comme le montre cette illustration.

Le didacticiel devrait prendre environ 20 minutes.

Ce didacticiel explique comment utiliser la JavaScript logique pour créer une solution adaptée à ce cas d'utilisation. Par exemple, vous apprendrez à lire une base de données pour déterminer quels employés ont atteint le premier anniversaire, à traiter les données et à envoyer un message texte à l'aide d'une fonction Lambda. Vous apprendrez ensuite à utiliser une expression cron pour appeler la fonction Lambda tous les jours de la semaine.

Ce AWS didacticiel utilise une table Amazon DynamoDB nommée Employee qui contient ces champs.

- id - la clé primaire de la table.
- FirstName : prénom de l'employé.
- téléphone - numéro de téléphone de l'employé.
- Date de début : date de début de l'employé.



> **Important**
>
> Coût de réalisation : Les AWS services inclus dans ce document sont inclus dans le niveau AWS gratuit. Cependant, veillez à désactiver toutes les ressources une fois que vous aurez terminé ce didacticiel pour vous assurer que vous n'êtes pas débité.

Pour créer l'application, procédez comme suit :

1. Compléter les prérequis
2. Créez les AWS ressources
3. Préparez le script du navigateur
4. Création et téléchargement de la fonction Lambda
5. Déployer la fonction Lambda
6. Exécutez l'application
7. Supprimer les ressources

Tâches préalables

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples Node.js et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions figurant sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

Créez les AWS ressources

Ce didacticiel nécessite les ressources suivantes.

- Une table Amazon DynamoDB nommée Employee avec une clé nommée Id et les champs illustrés dans l'illustration précédente. Assurez-vous de saisir les données correctes, y compris un téléphone portable valide avec lequel vous souhaitez tester ce cas d'utilisation. Pour plus d'informations, consultez la section Création d'une table.
- Rôle IAM associé à des autorisations permettant d'exécuter des fonctions Lambda.
- Un compartiment Amazon S3 pour héberger la fonction Lambda.

Vous pouvez créer ces ressources manuellement, mais nous vous recommandons de les approvisionner à l'aide de la méthode CloudFormation décrite dans ce didacticiel.

Créez les AWS ressources à l'aide de CloudFormation

CloudFormation vous permet de créer et de provisionner des déploiements AWS d'infrastructure de manière prévisible et répétée. Pour plus d'informations CloudFormation, consultez le guide de AWS CloudFormation l'utilisateur.

Pour créer la CloudFormation pile à l'aide de AWS CLI :

1. Installez et configurez en AWS CLI suivant les instructions du guide de l'AWS CLI utilisateur.
2. Créez un fichier nommé `setup.yaml` dans le répertoire racine du dossier de votre projet et GitHubcopiez-y le contenu.

> **Note**
>
> Le CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour plus d'informations à ce sujet AWS CDK, consultez le guide du AWS Cloud Development Kit (AWS CDK) développeur.

3. Exécutez la commande suivante depuis la ligne de commande, en la *STACK_NAME* remplaçant par un nom unique pour la pile.

   > **Important**
   >
   > Le nom de la pile doit être unique au sein d'une AWS région et d'un AWS compte. Vous pouvez spécifier jusqu'à 128 caractères. Les chiffres et les tirets sont autorisés.

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

   Pour plus d'informations sur les paramètres de `create-stack` commande, consultez le guide de référence des AWS CLI commandes et le guide de CloudFormation l'utilisateur.

   Consultez la liste des ressources dans la console en ouvrant la pile sur le CloudFormation tableau de bord et en choisissant l'onglet Ressources. Vous en avez besoin pour le didacticiel.

4. Lorsque la pile est créée, utilisez le AWS SDK pour JavaScript pour remplir la table DynamoDB, comme décrit dans. Renseignez la table DynamoDB

Renseignez la table DynamoDB

Pour remplir le tableau, créez d'abord un répertoire nommé`libs`, puis créez un fichier nommé`dynamoClient.js`, puis collez-y le contenu ci-dessous.

```
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 };
```

Ce code est disponible ici GitHub.

Créez ensuite un fichier nommé `populate-table.js` dans le répertoire racine du dossier de votre projet et GitHubcopiez-y le contenu. Pour l'un des articles, remplacez la valeur de la phone propriété par un numéro de téléphone portable valide au format E.164, et la valeur `startDate` par la date du jour.

Exécutez la commande suivante depuis la ligne de commande.

```
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();
```

Ce code est disponible ici GitHub.

Création de la AWS Lambda fonction

Configuration du kit SDK

Importez d'abord les modules et commandes AWS SDK pour JavaScript (v3) requis : `ScanCommand` DynamoDB `DynamoDBClient` et la commande Amazon `SNSClient` SNS. `PublishCommand` Remplacez *`REGION`* par la AWS région. Calculez ensuite la date du jour et attribuez-la à un paramètre. Créez ensuite les paramètres du `ScanCommand` fichier .Replace *`TABLE_NAME`* avec le nom de la table que vous avez créée dans la Créez les AWS ressources section de cet exemple.

L'extrait de code suivant illustre cette étape. (Pour obtenir l'exemple complet, consultez Regroupement de la fonction Lambda.)

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");
```

```
//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

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

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

Analyse de la table DynamoDB

Créez d'abord une async/await fonction appelée `sendText` pour publier un message texte à l'aide d'Amazon SNS`PublishCommand`. Ajoutez ensuite un schéma de `try` blocs qui analyse la table DynamoDB à la recherche des employés à l'occasion de leur anniversaire de travail aujourd'hui, puis appelle `sendText` la fonction pour envoyer un message texte à ces employés. En cas d'erreur, le `catch` bloc est appelé.

L'extrait de code suivant illustre cette étape. (Pour obtenir l'exemple complet, consultez Regroupement de la fonction Lambda.)

```
exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
```

```
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Regroupement de la fonction Lambda

Cette rubrique décrit comment regrouper `mylambdafunction.js` les AWS SDK pour JavaScript modules requis pour cet exemple dans un fichier groupé appelé`index.js`.

1. Si ce n'est pas déjà fait, suivez cet exemple Tâches préalables pour installer le webpack.

   > **Note**
   >
   > Pour plus d'informations sur Webpack, consultezRegroupez des applications avec Webpack.

2. Exécutez la commande suivante dans la ligne de commande JavaScript pour regrouper les informations de cet exemple dans un fichier appelé <`index.js`> :

   ```
   webpack mylamdbafunction.js --mode development --target node --devtool false --
   output-library-target umd -o index.js
   ```

> **Important**
>
> Notez que la sortie est nomméeindex.js. Cela est dû au fait que les fonctions Lambda doivent avoir un index.js gestionnaire pour fonctionner.

3. Compressez le fichier de sortie groupéindex.js,, dans un fichier ZIP nommémy-lambda-function.zip.
4. mylambdafunction.zipTéléchargez-le dans le compartiment Amazon S3 que vous avez créé dans la Créez les AWS ressources rubrique de ce didacticiel.

Voici le code de script de navigateur complet pourmylambdafunction.js.

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

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

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

```
// Create the client service objects.
const dbclient = new DynamoDBClient({ region: REGION });
const snsclient = new SNSClient({ region: REGION });

exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Déployez la fonction Lambda.

À la racine de votre projet, créez un `lambda-function-setup.js` fichier et collez-y le contenu ci-dessous.

*BUCKET_NAME*Remplacez-le par le nom du compartiment Amazon S3 dans lequel vous avez chargé la version ZIP de votre fonction Lambda. Remplacez *ZIP_FILE_NAME* par le nom la version ZIP de votre fonction Lambda. *IAM_ROLE_ARN*Remplacez-le par le numéro de ressource Amazon (ARN) du

rôle IAM que vous avez créé dans le Créez les AWS ressources sujet de ce didacticiel. Remplacez *LAMBDA_FUNCTION_NAME* par le nom de la fonction Lambda.

```
// Load the required Lambda client and commands.
const {
   CreateFunctionCommand,
} = require("@aws-sdk/client-lambda");
const {
   lambdaClient
} = require("..libs/lambdaClient.js");

// Instantiate an Lambda client service object.
const lambda = new LambdaClient({ region: REGION });

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
 Services (Amazon SNS) to " +
    "send employees an email the each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambda.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

Entrez ce qui suit sur la ligne de commande pour déployer la fonction Lambda.

```
node lambda-function-setup.js
```

Cet exemple de code est disponible ici GitHub.

Configurer CloudWatch pour appeler les fonctions Lambda

Pour configurer CloudWatch afin d'invoquer les fonctions Lambda, procédez comme suit :

1. Ouvrez la page Functions (Fonctions) sur la console Lambda.
2. Choisissez la fonction Lambda.
3. Sous Designer (Concepteur), choisissez Add trigger (Ajouter un déclencheur).
4. Définissez le type de déclencheur sur CloudWatch Evénements/ EventBridge.
5. Pour Règle, choisissez Créer une nouvelle règle.
6. Renseignez le nom et la description de la règle.
7. Pour le type de règle, sélectionnez Expression de planification.
8. Dans le champ Expression de planification, entrez une expression cron. Par exemple, cron (0) 12 ? * DU LUNDI AU VENDREDI (*).
9. Choisissez Ajouter.

> **Note**
>
> Pour plus d'informations, consultez la section Utilisation de Lambda avec des CloudWatch événements.

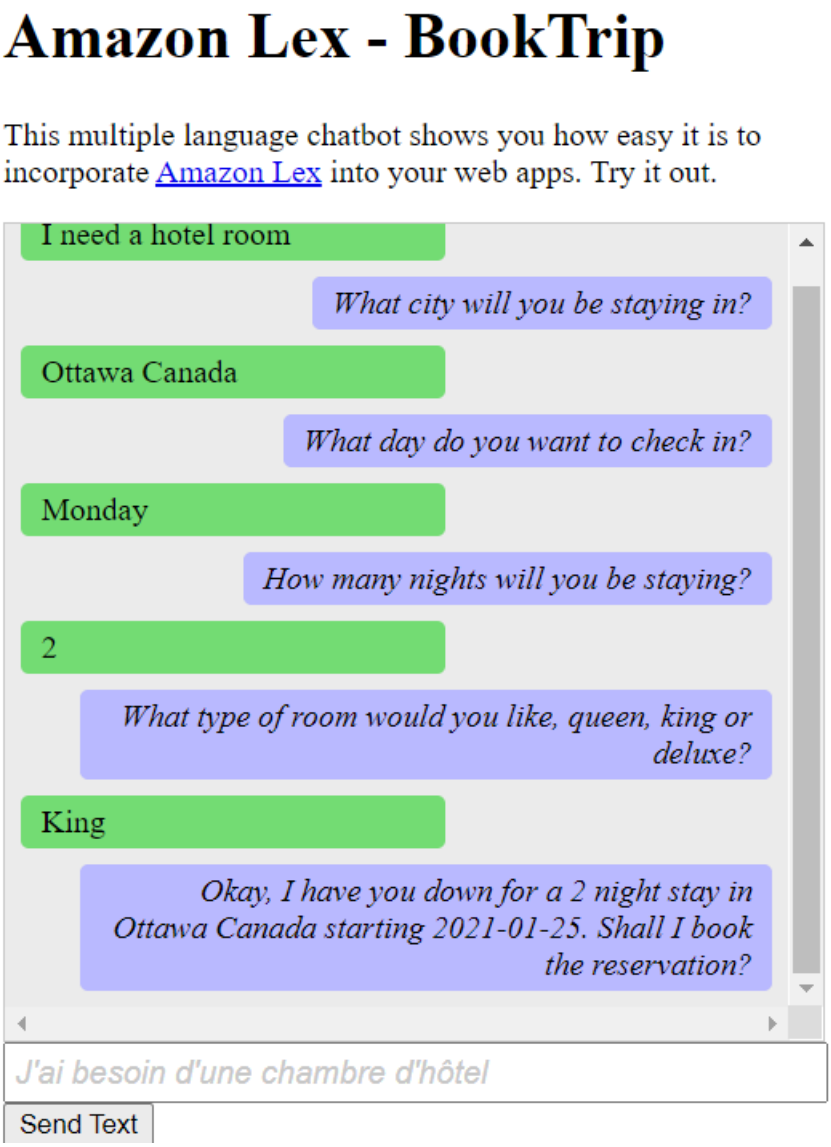
Supprimer les ressources

Félicitations ! Vous avez invoqué une fonction Lambda par le biais d'événements CloudWatch planifiés par Amazon à l'aide du. AWS SDK pour JavaScript Comme indiqué au début de ce didacticiel, veillez à désactiver toutes les ressources que vous créez pendant que vous suivez ce didacticiel pour vous assurer que vous n'êtes pas débité. Pour ce faire, supprimez la CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

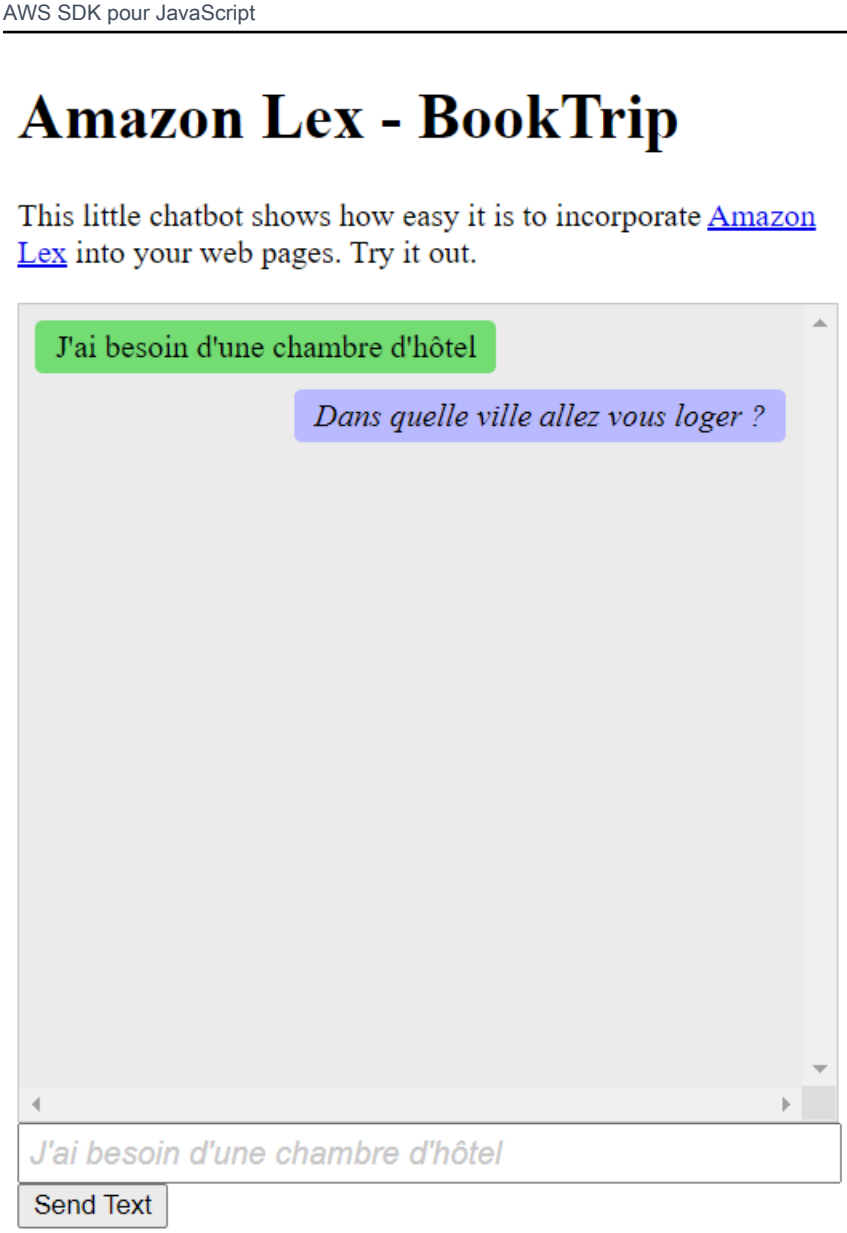
1. Ouvrez la CloudFormation console.
2. Sur la page Stacks, sélectionnez la pile.
3. Sélectionnez Supprimer.

Création d’un chatbot Amazon Lex

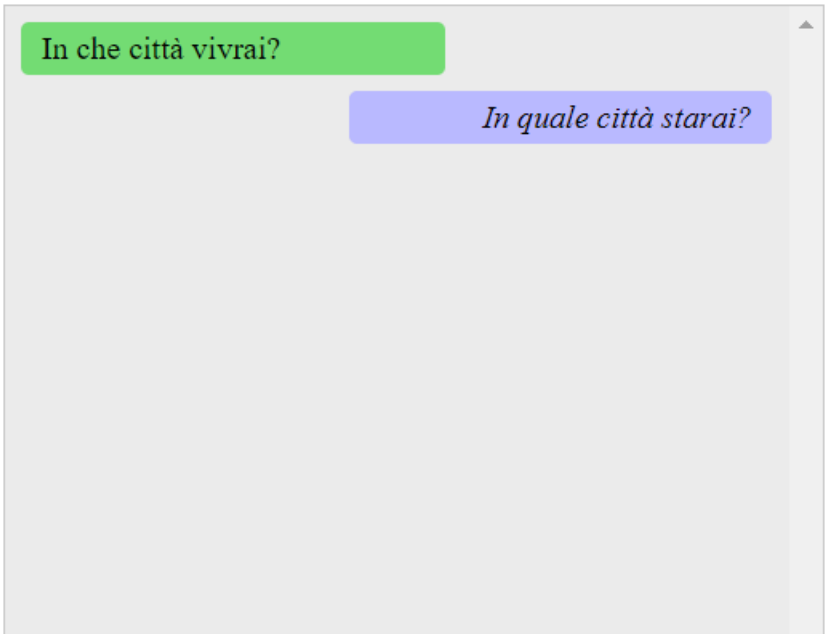
Vous pouvez créer un chatbot Amazon Lex au sein d'une application Web pour engager les visiteurs de votre site Web. Un chatbot Amazon Lex est une fonctionnalité qui permet de discuter en ligne avec les utilisateurs sans établir de contact direct avec une personne. Par exemple, l'illustration suivante montre un chatbot Amazon Lex qui invite un utilisateur à réserver une chambre d'hôtel.



Le chatbot Amazon Lex créé dans ce AWS didacticiel est capable de gérer plusieurs langues. Par exemple, un utilisateur qui parle français peut saisir du texte en français et obtenir une réponse en français.



De même, un utilisateur peut communiquer avec le chatbot Amazon Lex en italien.



Ce AWS didacticiel vous explique comment créer un chatbot Amazon Lex et comment l'intégrer dans une application Web Node.js. Le AWS SDK pour JavaScript (v3) est utilisé pour appeler les AWS services suivants :

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

Coût de réalisation : Les AWS services inclus dans ce document sont inclus dans le niveau AWS gratuit.

Remarque : veillez à désactiver toutes les ressources que vous créez pendant que vous suivez ce didacticiel pour vous assurer que vous n'êtes pas débité.

Pour créer l'application, procédez comme suit :

1. Conditions préalables
2. Allocation des ressources

3. Création d'un chatbot Amazon Lex
4. Créez le code HTML
5. Créez le script du navigateur
6. Étapes suivantes

Conditions préalables

Pour configurer et exécuter cet exemple, vous devez d'abord :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK pour JavaScript et tiers. Suivez les instructions figurant sur GitHub.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la fourniture d'un fichier d'informations d'identification partagé, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le guide de référence AWS SDKs et Tools.

> **Important**
>
> Cet exemple utilise ECMAScript6 (ES6). Cela nécessite la version 13.x ou supérieure de Node.js. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. .
> Toutefois, si vous préférez utiliser la syntaxe CommonJS, veuillez vous référer àJavaScript ES6Syntaxe /CommonJS.

Créez les AWS ressources

Ce didacticiel nécessite les ressources suivantes.

- Un rôle IAM non authentifié avec des autorisations associées pour :
  - Amazon Comprehend
  - Amazon Translate
  - Amazon Lex

Vous pouvez créer ces ressources manuellement, mais nous vous recommandons de les approvisionner en suivant les AWS CloudFormation instructions de ce didacticiel.

Créez les AWS ressources à l'aide de CloudFormation

CloudFormation vous permet de créer et de provisionner des déploiements AWS d'infrastructure de manière prévisible et répétée. Pour plus d'informations CloudFormation, consultez le guide de AWS CloudFormation l'utilisateur.

Pour créer la CloudFormation pile à l'aide de AWS CLI :

1. Installez et configurez en AWS CLI suivant les instructions du guide de l'AWS CLI utilisateur.
2. Créez un fichier nommé `setup.yaml` dans le répertoire racine du dossier de votre projet et GitHubcopiez-y le contenu.

   > **Note**
   >
   > Le CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour plus d'informations à ce sujet AWS CDK, consultez le guide du AWS Cloud Development Kit (AWS CDK) développeur.

3. Exécutez la commande suivante depuis la ligne de commande, en la *`STACK_NAME`* remplaçant par un nom unique pour la pile.

   > **Important**
   >
   > Le nom de la pile doit être unique au sein d'une AWS région et d'un AWS compte. Vous pouvez spécifier jusqu'à 128 caractères. Les chiffres et les tirets sont autorisés.

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

   Pour plus d'informations sur les paramètres de `create-stack` commande, consultez le guide de référence des AWS CLI commandes et le guide de CloudFormation l'utilisateur.

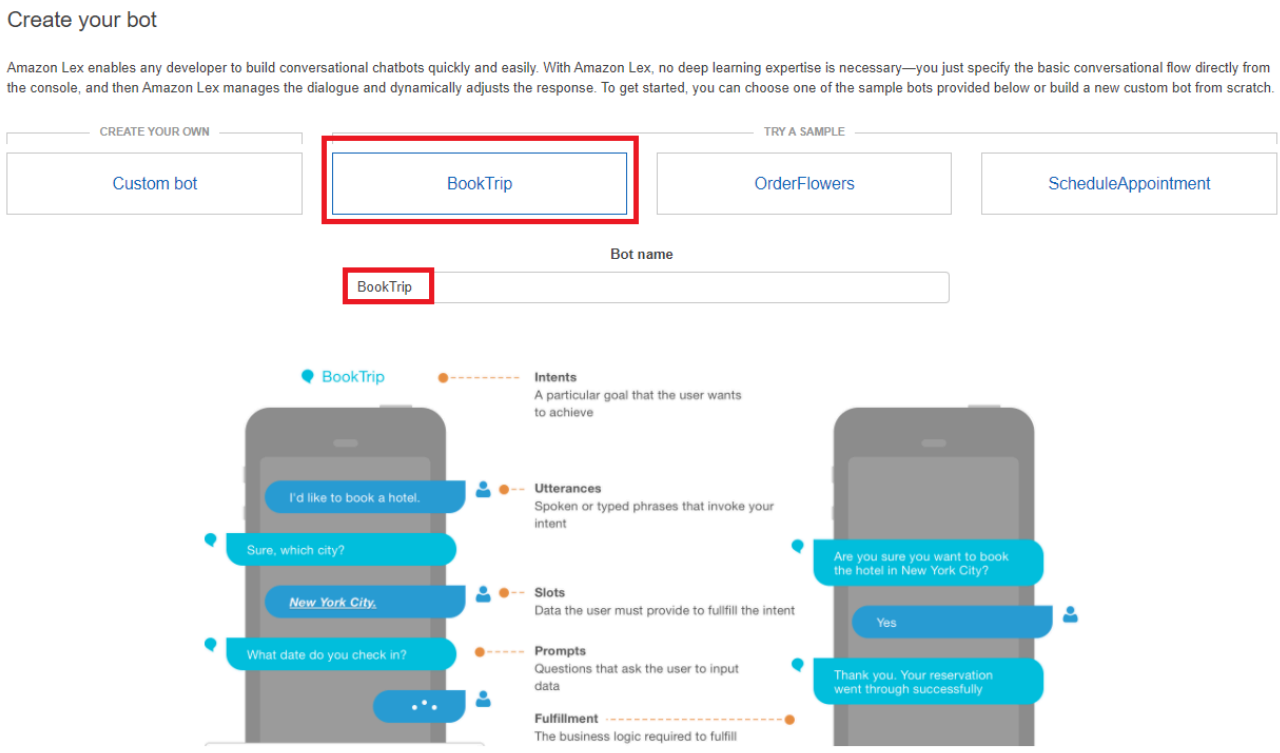
   Pour afficher les ressources créées, ouvrez la console Amazon Lex, choisissez la pile, puis sélectionnez l'onglet Ressources.

Créer un bot Amazon Lex

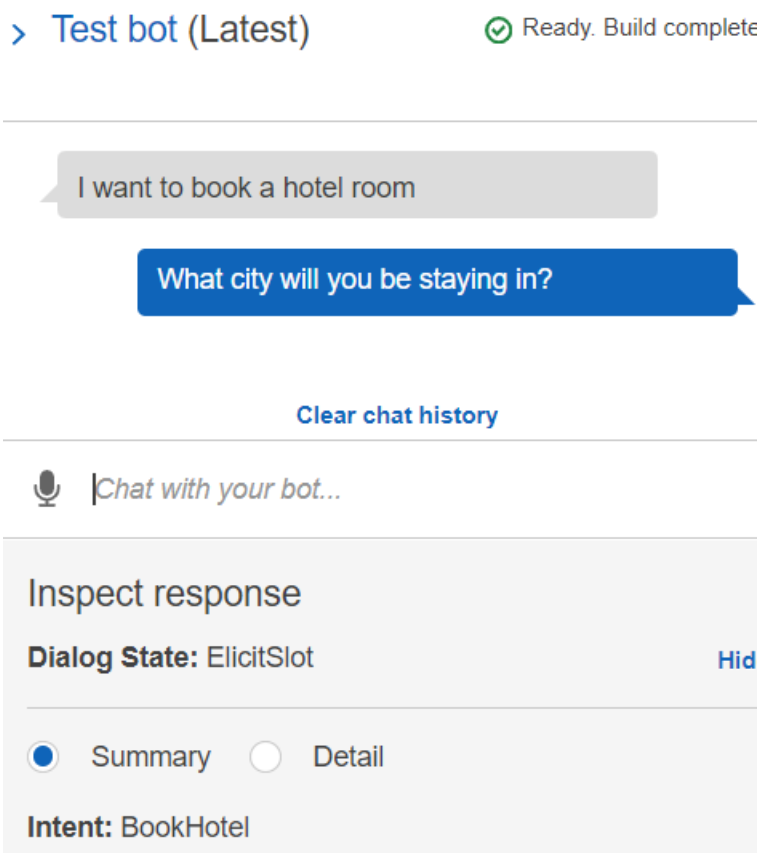
> **Important**
>
> Utilisez la version 1 de la console Amazon Lex pour créer le bot. Cet exemple ne fonctionne pas avec les robots créés à l'aide de la V2.

La première étape consiste à créer un chatbot Amazon Lex à l'aide de la console de gestion Amazon Web Services. Dans cet exemple, l'BookTripexemple Amazon Lex est utilisé. Pour plus d'informations, voir Book Trip.

- Connectez-vous à la console de gestion Amazon Web Services et ouvrez la console Amazon Lex sur la console Amazon Web Services.
- Sur la page Bots, choisissez Create.
- Choisissez BookTripBlueprint (conservez le nom du bot par défaut BookTrip).



- Renseignez les paramètres par défaut et choisissez Create (la console affiche le BookTripbot). Dans l'onglet Éditeur, passez en revue les détails des intentions préconfigurées.
- Testez le bot dans la fenêtre de test. Commencez le test en tapant Je souhaite réserver une chambre d'hôtel.



- Choisissez Publier et spécifiez un nom d'alias (vous aurez besoin de cette valeur pour utiliser le AWS SDK pour JavaScript).

> **Note**
>
> Vous devez faire référence au nom et à l'alias du bot dans votre JavaScript code.

Créez le code HTML

Créez un fichier nommé `index.html`. Copiez et collez le code ci-dessous dans`index.html`. Ce code HTML fait référence`main.js`. Il s'agit d'une version groupée de index.js, qui inclut les AWS SDK pour JavaScript modules requis. Vous allez créer ce fichier dansCréez le code HTML. `index.html`également des références`style.css`, ce qui ajoute les styles.

```
<!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>
```

Ce code est également disponible ici sur GitHub.

Créez le script du navigateur

Créez un fichier nommé `index.js`. Copiez et collez le code ci-dessous dans`index.js`. Importez les AWS SDK pour JavaScript modules et commandes requis. Créez des clients pour Amazon Lex, Amazon Comprehend et Amazon Translate. Remplacez *`REGION`* par AWS Region et *`IDENTITY_POOL_ID`* par l'ID du pool d'identités que vous avez créé dans leCréez les AWS ressources . Pour récupérer cet ID de pool d'identités, ouvrez le pool d'identités dans la console Amazon Cognito, choisissez Modifier le pool d'identités, puis choisissez Exemple de code dans le menu latéral. L'ID du pool d'identités est affiché en rouge dans la console.

Créez d'abord un `libs` répertoire et créez les objets clients de service requis en créant trois fichiers`comprehendClient.js`,`lexClient.js`, et`translateClient.js`. Collez le code

approprié ci-dessous dans chacun d'eux, puis remplacez *REGION* et *IDENTITY_POOL_ID* dans chaque fichier.

> **Note**
>
> Utilisez l'ID du pool d'identités Amazon Cognito dans lequel vous avez créé. Créez les AWS ressources à l'aide de CloudFormation

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

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

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

export { comprehendClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { LexRuntimeServiceClient } from "@aws-sdk/client-lex-runtime-service";

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

// Create an Amazon Lex service client object.
const lexClient = new LexRuntimeServiceClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
```

```
  }),
});

export { lexClient };
```

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

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

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

export { translateClient };
```

Ce code est disponible ici GitHub. .

Créez ensuite un `index.js` fichier et collez-y le code ci-dessous.

Remplacez *`BOT_ALIAS`* et *`BOT_NAME`* par l'alias et le nom de votre bot Amazon Lex, respectivement, et *`USER_ID`* par un identifiant utilisateur. La fonction `createResponse` asynchrone effectue les opérations suivantes :

- Prend le texte saisi par l'utilisateur dans le navigateur et utilise Amazon Comprehend pour déterminer son code de langue.
- Prend le code de langue et utilise Amazon Translate pour traduire le texte en anglais.
- Prend le texte traduit et utilise Amazon Lex pour générer une réponse.
- Publie la réponse sur la page du navigateur.

```
import { DetectDominantLanguageCommand } from "@aws-sdk/client-comprehend";
import { TranslateTextCommand } from "@aws-sdk/client-translate";
```

```
import { PostTextCommand } from "@aws-sdk/client-lex-runtime-service";
import { lexClient } from "./libs/lexClient.js";
import { translateClient } from "./libs/translateClient.js";
import { comprehendClient } from "./libs/comprehendClient.js";

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

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

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

  const lexTextResponse = lexResponse;

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

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

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

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

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

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

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

Ce code est disponible ici GitHub. .

Utilisez maintenant webpack pour regrouper les AWS SDK pour JavaScript modules `index.js` et dans un seul fichier,`main.js`.

1. Si ce n'est pas déjà fait, suivez cet exemple Conditions préalables pour installer le webpack.

   > **Note**
   >
   > Pour plus d'informations sur Webpack, consultezRegroupez des applications avec Webpack.

2. Exécutez la commande suivante dans la ligne de commande JavaScript pour regrouper les informations de cet exemple dans un fichier appelé `main.js` :

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

Étapes suivantes

Félicitations ! Vous avez créé une application Node.js qui utilise Amazon Lex pour créer une expérience utilisateur interactive. Comme indiqué au début de ce didacticiel, veillez à désactiver

toutes les ressources que vous créez pendant que vous suivez ce didacticiel pour vous assurer que vous n'êtes pas débité. Pour ce faire, supprimez la CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

1. Ouvrez la CloudFormation console.
2. Sur la page Stacks, sélectionnez la pile.
3. Sélectionnez Delete (Supprimer).

Pour d'autres exemples AWS interservices, consultez les exemples AWS SDK pour JavaScript interservices.

SDK pour exemples de JavaScript code (v3)

Les exemples de code présentés dans cette rubrique vous montrent comment utiliser le AWS SDK pour JavaScript (v3) avec AWS.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Certains services contiennent des exemples de catégories supplémentaires qui montrent comment tirer parti des bibliothèques ou des fonctions spécifiques au service.

Services

- Exemples d'API Gateway utilisant le SDK pour JavaScript (v3)
- Exemples d'Aurora utilisant le SDK pour JavaScript (v3)
- Exemples d'Auto Scaling utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Bedrock utilisant le SDK pour JavaScript (v3)
- Exemples d'exécution Amazon Bedrock utilisant le SDK pour JavaScript (v3)
- Exemples d'agents Amazon Bedrock utilisant le SDK pour JavaScript (v3)
- Exemples d'exécution d'Amazon Bedrock Agents utilisant le SDK pour JavaScript (v3)
- CloudWatch exemples d'utilisation du SDK pour JavaScript (v3)
- CloudWatch Exemples d'événements utilisant le SDK pour JavaScript (v3)
- CloudWatch Exemples de journaux utilisant le SDK pour JavaScript (v3)
- CodeBuild exemples d'utilisation du SDK pour JavaScript (v3)
- Exemples d'Amazon Cognito Identity utilisant le SDK pour JavaScript (v3)
- Exemples de fournisseurs d'identité Amazon Cognito utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Comprehend utilisant le SDK pour JavaScript (v3)

- Exemples d'Amazon DocumentDB utilisant le SDK pour JavaScript (v3)
- Exemples DynamoDB utilisant le SDK JavaScript pour (v3)
- EC2 Exemples Amazon utilisant le SDK pour JavaScript (v3)
- Elastic Load Balancing - Exemples de version 2 utilisant le SDK pour JavaScript (v3)
- Résolution des entités AWS exemples d'utilisation du SDK pour JavaScript (v3)
- EventBridge exemples d'utilisation du SDK pour JavaScript (v3)
- Exemples d'Amazon Glacier utilisant le SDK pour JavaScript (v3)
- AWS Glue exemples d'utilisation du SDK pour JavaScript (v3)
- HealthImaging exemples d'utilisation du SDK pour JavaScript (v3)
- Exemples d'IAM utilisant le SDK pour JavaScript (v3)
- AWS IoT SiteWise exemples d'utilisation du SDK pour JavaScript (v3)
- Exemples Kinesis utilisant le SDK pour JavaScript (v3)
- Exemples Lambda utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Lex utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Location utilisant le SDK pour JavaScript (v3)
- Exemples Amazon MSK utilisant le SDK pour JavaScript (v3)
- Exemples Amazon Personalize à l'aide du SDK pour JavaScript (v3)
- Exemples d'Amazon Personalize Events à l'aide du SDK pour JavaScript (v3)
- Exemples d'Amazon Personalize Runtime à l'aide du SDK pour JavaScript (v3)
- Exemples d'Amazon Pinpoint utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Polly utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon RDS utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon RDS Data Service utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Redshift utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Rekognition utilisant le SDK pour (v3) JavaScript
- Exemples d'Amazon S3 utilisant le SDK pour JavaScript (v3)
- SageMaker Exemples d'IA utilisant le SDK pour JavaScript (v3)
- Exemples de Secrets Manager utilisant le SDK pour JavaScript (v3)
- Exemples Amazon SES utilisant le SDK pour JavaScript (v3)
- Exemples Amazon SNS utilisant le SDK pour JavaScript (v3)

- Exemples Amazon SQS utilisant le SDK pour JavaScript (v3)
- Exemples de Step Functions utilisant le SDK pour JavaScript (v3)
- AWS STS exemples d'utilisation du SDK pour JavaScript (v3)
- Support exemples d'utilisation du SDK pour JavaScript (v3)
- Exemples de Systems Manager utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Textract utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Transcribe utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon Translate utilisant le SDK pour JavaScript (v3)

Exemples d'API Gateway utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la AWS SDK pour JavaScript version (v3) avec API Gateway.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Scénarios

Scénarios

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour JavaScript (v3)

> Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Utiliser API Gateway pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par Amazon API Gateway.

SDK pour JavaScript (v3)

Montre comment créer une AWS Lambda fonction à l'aide de l'API JavaScript d'exécution Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une fonction Lambda invoquée par Amazon API Gateway qui analyse une table Amazon DynamoDB à la recherche d'anniversaires professionnels et utilise Amazon Simple Notification Service (Amazon SNS) pour envoyer un message texte à vos employés qui les félicitent à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Exemples d'Aurora utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la AWS SDK pour JavaScript version (v3) avec Aurora.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Scénarios

Créer un outil de suivi des éléments de travail sans serveur Aurora

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une base de données Amazon Aurora sans serveur et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK pour JavaScript (v3) pour créer une application Web qui suit les éléments de travail dans une base de données Amazon Aurora et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise un front end créé avec React.js pour interagir avec un backend Express Node.js.

- Intégrez une application Web React.js à Services AWS.
- Lister, ajouter et mettre à jour des éléments dans une table Aurora.
- Envoyez un rapport par e-mail sur les éléments de travail filtrés en utilisant Amazon SES.
- Déployez et gérez des exemples de ressources à l'aide du AWS CloudFormation script inclus.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Aurora
- Amazon RDS
- Services de données Amazon RDS
- Amazon SES

Exemples d'Auto Scaling utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Auto Scaling.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions
- Scénarios

Actions

`AttachLoadBalancerTargetGroups`

L'exemple de code suivant montre comment utiliser`AttachLoadBalancerTargetGroups`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

- Pour plus de détails sur l'API, reportez-vous AttachLoadBalancerTargetGroupsà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer et gérer un service résilient

L'exemple de code suivant montre comment créer un service Web à charge équilibrée qui renvoie des recommandations de livres, de films et de chansons. L'exemple montre comment le service répond aux défaillances et comment le restructurer pour accroître la résilience en cas de défaillance.

- Utilisez un groupe Amazon EC2 Auto Scaling pour créer des instances Amazon Elastic Compute Cloud (Amazon EC2) sur la base d'un modèle de lancement et pour maintenir le nombre d'instances dans une plage spécifiée.
- Gérez et distribuez les requêtes HTTP avec Elastic Load Balancing.
- Surveillez l'état des instances d'un groupe Auto Scaling et transférez les demandes uniquement aux instances saines.
- Exécutez un serveur Web Python sur chaque EC2 instance pour gérer les requêtes HTTP. Le serveur Web répond par des recommandations et des surveillances de l'état.
- Simulez un service de recommandation avec une table Amazon DynamoDB.

- Contrôlez la réponse du serveur Web aux demandes et aux contrôles de santé en mettant à jour AWS Systems Manager les paramètres.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

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

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

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

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

/**
```

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

// Call function if run directly
import { fileURLToPath } from "node:url";

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

Créez des étapes pour déployer toutes les ressources.

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
```

```
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
```

```
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
```

```
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );
```

```
    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
```

```
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
```

```
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
```

```
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
```

```
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
```

```
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
```

```
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
```

```
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
```

```
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      }
      return MESSAGES.noIpRules;
    },
```

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

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    }
    return false;
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
```

```
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

Créez des étapes pour exécuter la démo.

```
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;
}
```

Créez des étapes pour déployer toutes les ressources.

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

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
```

```
import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
```

```
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
    return MESSAGES.deletedKeyPair.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

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

```
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
```

```
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
    return MESSAGES.deletedInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
```

```
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
    return MESSAGES.deletedInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
```

```
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
```

```
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
```

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

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

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

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

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

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

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

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

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

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript .
  - AttachLoadBalancerTargetGroups

- CreateAutoScalingGroup
- CreateInstanceProfile
- CreateLaunchTemplate
- CreateListener
- CreateLoadBalancer
- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Exemples d'Amazon Bedrock utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec Amazon Bedrock.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour Amazon Bedrock

L'exemple de code suivant montre comment commencer à utiliser Amazon Bedrock.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

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

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

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

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

  for (const model of models) {
    console.log("=".repeat(42));
    console.log(` Model: ${model.modelId}`);
    console.log("-".repeat(42));
    console.log(` Name: ${model.modelName}`);
    console.log(` Provider: ${model.providerName}`);
    console.log(` Model ARN: ${model.modelArn}`);
    console.log(` Input modalities: ${model.inputModalities}`);
    console.log(` Output modalities: ${model.outputModalities}`);
    console.log(` Supported customizations: ${model.customizationsSupported}`);
    console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
    console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
    console.log(`${"=".repeat(42)}\n`);
  }

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

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

  return response;
};

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

- Pour plus de détails sur l'API, reportez-vous ListFoundationModelsà la section Référence des AWS SDK pour JavaScript API.

Actions

`GetFoundationModel`

L'exemple de code suivant montre comment utiliser`GetFoundationModel`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez des détails sur un modèle de fondation.

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

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

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

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

  const response = await client.send(command);

  return response.modelDetails;
};
```

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

- Pour plus de détails sur l'API, reportez-vous GetFoundationModelà la section Référence des AWS SDK pour JavaScript API.

`ListFoundationModels`

L'exemple de code suivant montre comment utiliser`ListFoundationModels`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les modèles de fondation disponibles.

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

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

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

  const input = {
```

```
    // byProvider: 'STRING_VALUE',
    // byCustomizationType: 'FINE_TUNING' || 'CONTINUED_PRE_TRAINING',
    // byOutputModality: 'TEXT' || 'IMAGE' || 'EMBEDDING',
    // byInferenceType: 'ON_DEMAND' || 'PROVISIONED',
  };

  const command = new ListFoundationModelsCommand(input);

  const response = await client.send(command);

  return response.modelSummaries;
};

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

- Pour plus de détails sur l'API, reportez-vous ListFoundationModelsà la section Référence des AWS SDK pour JavaScript API.

Exemples d'exécution Amazon Bedrock utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Bedrock Runtime.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d’un même service ou combinés à d’autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l’exécution du code en contexte.

Rubriques

- Mise en route
- Scénarios

- Amazon Nova
- Amazon Nova Canvas
- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

Mise en route

Bonjour Amazon Bedrock

L'exemple de code suivant montre comment commencer à utiliser Amazon Bedrock.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

import { fileURLToPath } from "node:url";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
```

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

const AWS_REGION = "us-east-1";

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

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

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

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

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

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

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

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

```
    console.log("Haiku returned multiple responses:");
    console.log(responses);
  }

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

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

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Invocation de plusieurs modèles de fondation sur Amazon Bedrock

L'exemple de code suivant montre comment préparer et envoyer une invite à une variété de modèles en langage large (LLMs) sur Amazon Bedrock

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

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

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

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

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

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

```
);

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

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

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

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - InvokeModel
  - InvokeModelWithResponseStream

Utilisation de l'outil avec l'API Converse

L'exemple de code suivant montre comment créer une interaction typique entre une application, un modèle d'IA génératif et des outils connectés ou comment APIs arbitrer les interactions entre l'IA et le monde extérieur. Il utilise comme exemple la connexion d'une API de météorologie externe au modèle d'IA afin de fournir des informations météorologiques en temps réel en fonction des données saisies par l'utilisateur.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécution principale du flux de scénario. Ce scénario orchestre la conversation entre l'utilisateur, l'API Converse Amazon Bedrock et un outil météo.

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

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

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

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

const systemPrompt = [
  {
    text:
```

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

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

```
  } catch (error) {
    console.log("error ", error);
  }
}

// Sends the conversation, the system prompt, and the tool spec to Amazon Bedrock,
 and returns the response.
// param "messages" - The conversation history including the next message to send.
// return - The response from Amazon Bedrock.
async function SendConversationtoBedrock(messages) {
  const bedRockRuntimeClient = new BedrockRuntimeClient({
    region: "us-east-1",
  });
  try {
    const modelId = "amazon.nova-lite-v1:0";
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: messages,
        system: systemPrompt,
        toolConfig: tools_config,
      }),
    );
    return response;
  } catch (caught) {
    if (caught.name === "ModelNotReady") {
      console.log(
        "`${caught.name}` - Model not ready, please wait and try again.",
      );
      throw caught;
    }
    if (caught.name === "BedrockRuntimeException") {
      console.log(
        '`${caught.name}` - "Error occurred while sending Converse request.',
      );
      throw caught;
    }
  }
}

// Processes the response received via Amazon Bedrock and performs the necessary
 actions based on the stop reason.
// param "response" - The model's response returned via Amazon Bedrock.
// param "messages" - The conversation history.
```

```
// param "max_recursions" - The maximum number of recursive calls allowed.
async function ProcessModelResponseAsync(response, messages, max_recursions) {
  if (max_recursions <= 0) {
    await HandleToolUseAsync(response, messages);
  }
  if (response.stopReason === "tool_use") {
    await HandleToolUseAsync(response, messages, max_recursions - 1);
  }
  if (response.stopReason === "end_turn") {
    const messageToPrint = response.output.message.content[0].text;
    console.log(messageToPrint.replace(/<[^>]+>/g, ""));
  }
}
// Handles the tool use case by invoking the specified tool and sending the tool's
 response back to Bedrock.
// The tool response is appended to the conversation, and the conversation is sent
 back to Amazon Bedrock for further processing.
// param "response" - the model's response containing the tool use request.
// param "messages" - the conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function HandleToolUseAsync(response, messages, max_recursions) {
  const toolResultFinal = [];
  try {
    const output_message = response.output.message;
    messages.push(output_message);
    const toolRequests = output_message.content;
    const toolMessage = toolRequests[0].text;
    console.log(toolMessage.replace(/<[^>]+>/g, ""));
    for (const toolRequest of toolRequests) {
      if (Object.hasOwn(toolRequest, "toolUse")) {
        const toolUse = toolRequest.toolUse;
        const latitude = toolUse.input.latitude;
        const longitude = toolUse.input.longitude;
        const toolUseID = toolUse.toolUseId;
        console.log(
          `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
        );
        if (toolUse.name === "Weather_Tool") {
          try {
            const current_weather = await callWeatherTool(
              longitude,
              latitude,
            ).then((current_weather) => current_weather);
            const currentWeather = current_weather;
```

```
            const toolResult = {
              toolResult: {
                toolUseId: toolUseID,
                content: [{ json: currentWeather }],
              },
            };
            toolResultFinal.push(toolResult);
          } catch (err) {
            console.log("An error occurred. ", err);
          }
        }
      }

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

  // Fetch the weather data.
  return fetch(apiUrl)
    .then((response) => {
      return response.json().then((current_weather) => {
        return current_weather;
      });
    })
    .catch((error) => {
```

```
      console.error("Error fetching weather data:", error);
    });
}
/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "input",
  default: "",
});

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

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

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

```
  },
);

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

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

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

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

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

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

const myScenario = new Scenario("Converse Tool Scenario", [
  greet,
  pressEnter,
  displayAskQuestion1,
  askQuestion1,
  pressEnter,
  displayAskQuestion2,
  askQuestion2,
  pressEnter,
  displayAskQuestion3,
  askQuestion3,
  pressEnter,
  displayAskQuestion4,
  askQuestion4,
  pressEnter,
  goodbye,
]);

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

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

```
}
```

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour JavaScript .

Amazon Nova

Converse

L'exemple de code suivant montre comment envoyer un message texte à Amazon Nova à l'aide de l'API Converse de Bedrock.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Amazon Nova à l'aide de l'API Converse de Bedrock.

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

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

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

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

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

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

// Step 5: Send and process the request
// - Send the request to the model
// - Extract and return the generated text from the response
try {
  const response = await client.send(new ConverseCommand(request));
  console.log(response.output.message.content[0].text);
} catch (error) {
```

```
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  throw error;
}
```

Envoyez une conversation de messages à Amazon Nova à l'aide de l'API Converse de Bedrock avec une configuration d'outil.

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

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

// Step 1: Create the Amazon Bedrock runtime client

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

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

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

// The system prompt to help Amazon Bedrock craft it's response.
const system_prompt = [
  {
    text:
```

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

```
      },
    },
  ],
};

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

// 3. Send the request to Amazon Bedrock, and returns the response.
export async function SendConversationtoBedrock(
  modelId,
  message,
  system_prompt,
  tool_config,
) {
  try {
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: message,
        system: system_prompt,
        toolConfig: tool_config,
      }),
    );
    if (response.stopReason === "tool_use") {
      const toolResultFinal = [];
      try {
        const output_message = response.output.message;
        message.push(output_message);
        const toolRequests = output_message.content;
        const toolMessage = toolRequests[0].text;
        console.log(toolMessage.replace(/<[^>]+>/g, ""));
        for (const toolRequest of toolRequests) {
          if (Object.hasOwn(toolRequest, "toolUse")) {
```

```
            const toolUse = toolRequest.toolUse;
            const sign = toolUse.input.sign;
            const toolUseID = toolUse.toolUseId;
            console.log(
              `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
            );
            if (toolUse.name === "top_song") {
              const toolResult = [];
              try {
                const top_song = await get_top_song(toolUse.input.sign).then(
                  (top_song) => top_song,
                );
                const toolResult = {
                  toolResult: {
                    toolUseId: toolUseID,
                    content: [
                      {
                        json: { song: top_song.song, artist: top_song.artist },
                      },
                    ],
                  },
                };
                toolResultFinal.push(toolResult);
              } catch (err) {
                const toolResult = {
                  toolUseId: toolUseID,
                  content: [{ json: { text: err.message } }],
                  status: "error",
                };
              }
            }
          }
        }
        const toolResultMessage = {
          role: "user",
          content: toolResultFinal,
        };
        // Step 4. Add the tool response to the conversation, and send it back to
Amazon Bedrock.

        message.push(toolResultMessage);
        await SendConversationtoBedrock(
          modelId,
          message,
```

```
          system_prompt,
          tool_config,
        );
      } catch (caught) {
        console.error(`${caught.message}`);
        throw caught;
      }
    }

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

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour JavaScript .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Amazon Nova à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Amazon Nova à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

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

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

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

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

// Step 3: Create the message
// The message includes the text prompt and specifies that it comes from the user
```

```
const inputText =
  "Describe the purpose of a 'hello world' program in one paragraph";
const message = {
  content: [{ text: inputText }],
  role: ConversationRole.USER,
};

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

// Step 5: Send and process the streaming request
// - Send the request to the model
// - Process each chunk of the streaming response
try {
  const response = await client.send(new ConverseStreamCommand(request));

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

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour JavaScript API.

Scénario : utilisation de l'outil avec l'API Converse

L'exemple de code suivant montre comment créer une interaction typique entre une application, un modèle d'IA génératif et des outils connectés ou comment APIs arbitrer les interactions entre l'IA et le monde extérieur. Il utilise comme exemple la connexion d'une API de météorologie externe au modèle d'IA afin de fournir des informations météorologiques en temps réel en fonction des données saisies par l'utilisateur.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécution principale du flux de scénario. Ce scénario orchestre la conversation entre l'utilisateur, l'API Converse Amazon Bedrock et un outil météo.

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

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

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

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

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

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

/// Starts the conversation with the user and handles the interaction with Bedrock.
async function askQuestion(userMessage) {
```

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

// Sends the conversation, the system prompt, and the tool spec to Amazon Bedrock,
 and returns the response.
// param "messages" - The conversation history including the next message to send.
// return - The response from Amazon Bedrock.
async function SendConversationtoBedrock(messages) {
  const bedRockRuntimeClient = new BedrockRuntimeClient({
    region: "us-east-1",
  });
  try {
    const modelId = "amazon.nova-lite-v1:0";
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: messages,
        system: systemPrompt,
        toolConfig: tools_config,
      }),
    );
    return response;
  } catch (caught) {
    if (caught.name === "ModelNotReady") {
      console.log(
        "`${caught.name}` - Model not ready, please wait and try again.",
      );
      throw caught;
    }
    if (caught.name === "BedrockRuntimeException") {
```

```
      console.log(
        ''${caught.name}` - "Error occurred while sending Converse request.',
      );
      throw caught;
    }
  }
}

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

```
        const longitude = toolUse.input.longitude;
        const toolUseID = toolUse.toolUseId;
        console.log(
          `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
        );
        if (toolUse.name === "Weather_Tool") {
          try {
            const current_weather = await callWeatherTool(
              longitude,
              latitude,
            ).then((current_weather) => current_weather);
            const currentWeather = current_weather;
            const toolResult = {
              toolResult: {
                toolUseId: toolUseID,
                content: [{ json: currentWeather }],
              },
            };
            toolResultFinal.push(toolResult);
          } catch (err) {
            console.log("An error occurred. ", err);
          }
        }
      }
    }

    const toolResultMessage = {
      role: "user",
      content: toolResultFinal,
    };
    messages.push(toolResultMessage);
    // Send the conversation to Amazon Bedrock
    await ProcessModelResponseAsync(
      await SendConversationtoBedrock(messages),
      messages,
    );
  } catch (error) {
    console.log("An error occurred. ", error);
  }
}
// Call the Weathertool.
// param = longitude of location
// param = latitude of location
async function callWeatherTool(longitude, latitude) {
```

```
  // Open-Meteo API endpoint
  const apiUrl = `https://api.open-meteo.com/v1/forecast?latitude=
${latitude}&longitude=${longitude}&current_weather=true`;

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

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

  { header: true },
```

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

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

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

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

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

const displayAskQuestion4 = new ScenarioOutput(
  "displayAskQuestion4",
```

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

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

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

const myScenario = new Scenario("Converse Tool Scenario", [
  greet,
  pressEnter,
  displayAskQuestion1,
  askQuestion1,
  pressEnter,
  displayAskQuestion2,
  askQuestion2,
  pressEnter,
  displayAskQuestion3,
  askQuestion3,
  pressEnter,
  displayAskQuestion4,
  askQuestion4,
  pressEnter,
  goodbye,
]);

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

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

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour JavaScript .

Amazon Nova Canvas

InvokeModel

L'exemple de code suivant montre comment invoquer Amazon Nova Canvas sur Amazon Bedrock pour générer une image.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une image avec Amazon Nova Canvas.

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

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

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

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

  // Step 3: Configure the request payload
  // First, set the main parameters:
  // - prompt: Text description of the image to generate
  // - seed: Random number for reproducible generation (0 to 858,993,459)
  const prompt = "A stylized picture of a cute old steampunk robot";
  const seed = Math.floor(Math.random() * 858993460);

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

```
      quality: "standard",
    },
  };

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

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

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

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK pour JavaScript API.

Anthropic Claude

Converse

L'exemple de code suivant montre comment envoyer un message texte à Anthropic Claude à l'aide de l'API Converse de Bedrock.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Anthropic Claude à l'aide de l'API Converse de Bedrock.

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

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

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

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

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

```
];

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

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

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

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour JavaScript .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Anthropic Claude à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Anthropic Claude à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// 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);
```

```
}
```

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour JavaScript API.

InvokeModel

L'exemple de code suivant montre comment envoyer un message texte à Anthropic Claude à l'aide de l'API Invoke Model.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez l'API Invoke Model pour envoyer un message texte.

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

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

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

```
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

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

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

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

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

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

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

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

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
```

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

  let completeMessage = "";

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

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

  // Return the final response
  return completeMessage;
};

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

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

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK pour JavaScript API.

InvokeModelWithResponseStream

L'exemple de code suivant montre comment envoyer un message texte aux modèles Anthropic Claude à l'aide de l'API Invoke Model et imprimer le flux de réponses.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez l'API Invoke Model pour envoyer un message texte et traiter le flux de réponses en temps réel.

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

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

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

```
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

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

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

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

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

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

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

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

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

  let completeMessage = "";

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

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

  // Return the final response
  return completeMessage;
};

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

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

- Pour plus de détails sur l'API, reportez-vous InvokeModelWithResponseStreamà la section Référence des AWS SDK pour JavaScript API.

Cohere Command

Converse

L'exemple de code suivant montre comment envoyer un message texte à Cohere Command à l'aide de l'API Converse de Bedrock.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Cohere Command à l'aide de l'API Converse de Bedrock.

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

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

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

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

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

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

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

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

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

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour JavaScript .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Cohere Command à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Cohere Command à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// 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);
}
```

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour JavaScript API.

Meta Llama

Converse

L'exemple de code suivant montre comment envoyer un message texte à Meta Llama à l'aide de l'API Converse de Bedrock.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Meta Llama à l'aide de l'API Converse de Bedrock.

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

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

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

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

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

```
];

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

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

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

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour JavaScript .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Meta Llama à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Meta Llama à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// 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);
```

```
}
```

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour JavaScript API.

InvokeModel

L'exemple de code suivant montre comment envoyer un message texte à Meta Llama, à l'aide de l'API Invoke Model.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez l'API Invoke Model pour envoyer un message texte.

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

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

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

// Set the model ID, e.g., Llama 3 70B Instruct.
const modelId = "meta.llama3-70b-instruct-v1:0";

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

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|><|start_header_id|>user<|end_header_id|>
```

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

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

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

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

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

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

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK pour JavaScript API.

InvokeModelWithResponseStream

L’exemple de code suivant montre comment envoyer un message texte à Meta Llama à l’aide de l’API Invoke Model et imprimer le flux de réponses.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez l'API Invoke Model pour envoyer un message texte et traiter le flux de réponses en temps réel.

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

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

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

// Set the model ID, e.g., Llama 3 70B Instruct.
const modelId = "meta.llama3-70b-instruct-v1:0";

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

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

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

```
  top_p: 0.9,
};

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

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

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

- Pour plus de détails sur l'API, reportez-vous InvokeModelWithResponseStreamà la section Référence des AWS SDK pour JavaScript API.

Mistral AI

Converse

L’exemple de code suivant montre comment envoyer un message texte à Mistral à l’aide de l’API Converse de Bedrock.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Mistral à l'aide de l'API Converse de Bedrock.

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

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

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

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

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

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

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

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

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour JavaScript .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Mistral à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Mistral à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// 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);
}
```

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour JavaScript API.

InvokeModel

L’exemple de code suivant montre comment envoyer un message texte aux modèles Mistral, à l’aide de l’API Invoke Model.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez l'API Invoke Model pour envoyer un message texte.

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

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

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

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

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

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

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

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

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

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

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

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK pour JavaScript API.

Exemples d'agents Amazon Bedrock utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec Amazon Bedrock Agents.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Actions

Mise en route

Bonjour agents Amazon Bedrock

L'exemple de code suivant montre comment démarrer avec les agents Amazon Bedrock.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

import {
```

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

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

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 region.
 * It then retrieves a list of existing agents using the streamlined paginator
 approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
 * - Use of the Bedrock Agents client to initialize and communicate with the AWS
 service.
 * - Listing resources in a paginated response pattern.
 * - Accessing an individual resource using a command object.
 *
 * @returns {Promise<void>} A promise that resolves when the function has completed
 execution.
 */
export const main = async () => {
  const region = "us-east-1";

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

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

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

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

```
  }

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

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

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

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

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

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - GetAgent
  - ListAgents

Actions

`CreateAgent`

L'exemple de code suivant montre comment utiliser`CreateAgent`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un agent.

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

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

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

  const command = new CreateAgentCommand({
    agentName,
    foundationModel,
    agentResourceRoleArn,
```

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

  return response.agent;
};

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

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

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

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

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

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

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

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

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

- Pour plus de détails sur l'API, reportez-vous CreateAgentà la section Référence des AWS SDK pour JavaScript API.

`DeleteAgent`

L'exemple de code suivant montre comment utiliser`DeleteAgent`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez un agent.

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

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

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

// Invoke main function if this file was run directly.
```

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

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

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

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

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

- Pour plus de détails sur l'API, reportez-vous DeleteAgentà la section Référence des AWS SDK pour JavaScript API.

`GetAgent`

L'exemple de code suivant montre comment utiliser`GetAgent`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez un agent.

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

import {
  BedrockAgentClient,
  GetAgentCommand,
```

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

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

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

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

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

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

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

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

- Pour plus de détails sur l'API, reportez-vous GetAgentà la section Référence des AWS SDK pour JavaScript API.

`ListAgentActionGroups`

L'exemple de code suivant montre comment utiliser`ListAgentActionGroups`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les groupes d'actions d'un agent.

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

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

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  // Create a paginator configuration
  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
```

```
  };

  const params = { agentId, agentVersion };

  const pages = paginateListAgentActionGroups(paginatorConfig, params);

  // Paginate until there are no more results
  const actionGroupSummaries = [];
  for await (const page of pages) {
    actionGroupSummaries.push(...page.actionGroupSummaries);
  }

  return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const actionGroupSummaries = [];
  do {
    const command = new ListAgentActionGroupsCommand({
      agentId,
      agentVersion,
      nextToken,
```

```
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
    const response = await client.send(command);

    for (const actionGroup of response.actionGroupSummaries || []) {
      actionGroupSummaries.push(actionGroup);
    }

    nextToken = response.nextToken;
  } while (nextToken);

  return actionGroupSummaries;
};

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

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

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
 'DRAFT').
  const agentVersion = "[DRAFT]";

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

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using ListAgentActionGroupsCommand:",
  );

  for (const actionGroup of await listAgentActionGroupsWithCommandObject(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }
```

```
  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using the paginateListAgents function:",
  );
  for (const actionGroup of await listAgentActionGroupsWithPaginator(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListAgentActionGroupsà la section Référence des AWS SDK pour JavaScript API.

`ListAgents`

L'exemple de code suivant montre comment utiliser`ListAgents`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les agents associés à un compte.

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

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

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 function.
```

```
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 ListAgentsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
```

```
  const agentSummaries = [];
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
    const paginatedResponse = await client.send(command);

    agentSummaries.push(...(paginatedResponse.agentSummaries || []));

    nextToken = paginatedResponse.nextToken;
  } while (nextToken);

  return agentSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
  console.log("Listing agents using ListAgentsCommand:");
  for (const agent of await listAgentsWithCommandObject()) {
    console.log(agent);
  }

  console.log("=".repeat(68));
  console.log("Listing agents using the paginateListAgents function:");
  for (const agent of await listAgentsWithPaginator()) {
    console.log(agent);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListAgentsà la section Référence des AWS SDK pour JavaScript API.

Exemples d'exécution d'Amazon Bedrock Agents utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec Amazon Bedrock Agents Runtime.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Actions

`InvokeAgent`

L'exemple de code suivant montre comment utiliser`InvokeAgent`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  BedrockAgentRuntimeClient,
  InvokeAgentCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * @typedef {Object} ResponseBody
```

```
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
  // });

  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (const chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
      console.log(chunk);
      const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
      completion += decodedResponse;
    }
```

```
    return { sessionId: sessionId, completion };
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeAgentà la section Référence des AWS SDK pour JavaScript API.

`InvokeFlow`

L'exemple de code suivant montre comment utiliser`InvokeFlow`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

import {
  BedrockAgentRuntimeClient,
  InvokeFlowCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * Invokes an alias of a flow to run the inputs that you specify and return
 * the output of each node as a stream.
```

```
 *
 * @param {{
 *  flowIdentifier: string,
 *  flowAliasIdentifier: string,
 *  prompt?: string,
 *  region?: string
 * }} options
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").FlowNodeOutput>} An
 object containing information about the output from flow invocation.
 */
export const invokeBedrockFlow = async ({
  flowIdentifier,
  flowAliasIdentifier,
  prompt = "Hi, how are you?",
  region = "us-east-1",
}) => {
  const client = new BedrockAgentRuntimeClient({ region });

  const command = new InvokeFlowCommand({
    flowIdentifier,
    flowAliasIdentifier,
    inputs: [
      {
        content: {
          document: prompt,
        },
        nodeName: "FlowInputNode",
        nodeOutputName: "document",
      },
    ],
  });

  let flowResponse = {};
  const response = await client.send(command);

  for await (const chunkEvent of response.responseStream) {
    const { flowOutputEvent, flowCompletionEvent } = chunkEvent;

    if (flowOutputEvent) {
      flowResponse = { ...flowResponse, ...flowOutputEvent };
      console.log("Flow output event:", flowOutputEvent);
    } else if (flowCompletionEvent) {
      flowResponse = { ...flowResponse, ...flowCompletionEvent };
      console.log("Flow completion event:", flowCompletionEvent);
```

```
    }
  }

  return flowResponse;
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    flowIdentifier: {
      type: "string",
      required: true,
    },
    flowAliasIdentifier: {
      type: "string",
      required: true,
    },
    prompt: {
      type: "string",
    },
    region: {
      type: "string",
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    invokeBedrockFlow(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeFlowà la section Référence des AWS SDK pour JavaScript API.

CloudWatch exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec CloudWatch.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`DeleteAlarms`

L'exemple de code suivant montre comment utiliser`DeleteAlarms`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";
```

```
const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteAlarmsà la section Référence des AWS SDK pour JavaScript API.

`DescribeAlarmsForMetric`

L'exemple de code suivant montre comment utiliser`DescribeAlarmsForMetric`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeAlarmsForMetricà la section Référence des AWS SDK pour JavaScript API.

`DisableAlarmActions`

L'exemple de code suivant montre comment utiliser`DisableAlarmActions`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DisableAlarmActionsà la section Référence des AWS SDK pour JavaScript API.

`EnableAlarmActions`

L'exemple de code suivant montre comment utiliser`EnableAlarmActions`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous EnableAlarmActionsà la section Référence des AWS SDK pour JavaScript API.

`ListMetrics`

L'exemple de code suivant montre comment utiliser`ListMetrics`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import {
  CloudWatchServiceException,
  ListMetricsCommand,
} from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = async () => {
  // Use the AWS console to see available namespaces and metric names. Custom
 metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
    Dimensions: [
      {
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

  try {
    const response = await client.send(command);
    console.log(`Metrics count: ${response.Metrics?.length}`);
    return response;
  } catch (caught) {
    if (caught instanceof CloudWatchServiceException) {
      console.error(`Error from CloudWatch. ${caught.name}: ${caught.message}`);
    } else {
```

```
      throw caught;
    }
  }
};
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListMetricsà la section Référence des AWS SDK pour JavaScript API.

`PutMetricAlarm`

L'exemple de code suivant montre comment utiliser`PutMetricAlarm`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
```

```
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
 the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutMetricAlarmà la section Référence des AWS SDK pour JavaScript API.

`PutMetricData`

L'exemple de code suivant montre comment utiliser`PutMetricData`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};
```

```
export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutMetricDataà la section Référence des AWS SDK pour JavaScript API.

CloudWatch Exemples d'événements utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec des CloudWatch événements.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`PutEvents`

L'exemple de code suivant montre comment utiliser`PutEvents`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
        Source: "my.app",

        // The name of the event that is being sent.
        DetailType: "My Custom Event",

        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutEventsà la section Référence des AWS SDK pour JavaScript API.

`PutRule`

L'exemple de code suivant montre comment utiliser`PutRule`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  // Request parameters for PutRule.
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/
API_PutRule.html#API_PutRule_RequestParameters
  const command = new PutRuleCommand({
    Name: process.env.CLOUDWATCH_EVENTS_RULE,

    // The event pattern for the rule.
    //  Example: {"source": ["my.app"]}
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,

    // The state of the rule. Valid values: ENABLED, DISABLED
    State: "ENABLED",
  });
```

```
  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutRuleà la section Référence des AWS SDK pour JavaScript API.

`PutTargets`

L'exemple de code suivant montre comment utiliser`PutTargets`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutTargetsCommand({
```

```
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,

    // The targets to add to the rule.
    Targets: [
      {
        Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
        // The ID of the target. Choose a unique ID for each target.
        Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutTargetsà la section Référence des AWS SDK pour JavaScript API.

CloudWatch Exemples de journaux utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec CloudWatch des journaux.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions
- Scénarios

Actions

`CreateLogGroup`

L'exemple de code suivant montre comment utiliser`CreateLogGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
```

```
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Pour plus de détails sur l'API, reportez-vous CreateLogGroupà la section Référence des AWS SDK pour JavaScript API.

`DeleteLogGroup`

L'exemple de code suivant montre comment utiliser`DeleteLogGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Pour plus de détails sur l'API, reportez-vous DeleteLogGroupà la section Référence des AWS SDK pour JavaScript API.

`DeleteSubscriptionFilter`

L'exemple de code suivant montre comment utiliser`DeleteSubscriptionFilter`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Pour plus de détails sur l'API, reportez-vous DeleteSubscriptionFilterà la section Référence des AWS SDK pour JavaScript API.

`DescribeLogGroups`

L'exemple de code suivant montre comment utiliser`DescribeLogGroups`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups?.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }

  console.log(logGroups);
  return logGroups;
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeLogGroupsà la section Référence des AWS SDK pour JavaScript API.

`DescribeSubscriptionFilters`

L'exemple de code suivant montre comment utiliser`DescribeSubscriptionFilters`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-
logs";
import { client } from "../libs/client.js";

const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Pour plus de détails sur l'API, reportez-vous DescribeSubscriptionFiltersà la section Référence des AWS SDK pour JavaScript API.

`GetQueryResults`

L'exemple de code suivant montre comment utiliser`GetQueryResults`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

- Pour plus de détails sur l'API, reportez-vous GetQueryResultsà la section Référence des AWS SDK pour JavaScript API.

`PutSubscriptionFilter`

L'exemple de code suivant montre comment utiliser`PutSubscriptionFilter`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
    // An ARN of a same-account Kinesis stream, Kinesis Firehose
    // delivery stream, or Lambda function.
```

```
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
SubscriptionFilters.html
    destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,

    // A name for the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

    // A filter pattern for subscribing to a filtered stream of log events.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
FilterAndPatternSyntax.html
    filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

    // The name of the log group. Messages in this group matching the filter pattern
    // will be sent to the destination ARN.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Pour plus de détails sur l'API, reportez-vous PutSubscriptionFilterà la section Référence des AWS SDK pour JavaScript API.

`StartLiveTail`

L'exemple de code suivant montre comment utiliser`StartLiveTail`.

SDK pour JavaScript (v3)

Joignez les fichiers requis.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-
cloudwatch-logs";
```

Gérez les événements de la session Live Tail.

```
async function handleResponseAsync(response) {
    try {
      for await (const event of response.responseStream) {
        if (event.sessionStart !== undefined) {
          console.log(event.sessionStart);
        } else if (event.sessionUpdate !== undefined) {
          for (const logEvent of event.sessionUpdate.sessionResults) {
            const timestamp = logEvent.timestamp;
            const date = new Date(timestamp);
            console.log("[" + date + "] " + logEvent.message);
          }
        } else {
            console.error("Unknown event type");
        }
      }
    } catch (err) {
        // On-stream exceptions are captured here
        console.error(err)
    }
}
```

Démarrez une session Live Tail.

```
    const client = new CloudWatchLogsClient();

    const command = new StartLiveTailCommand({
        logGroupIdentifiers: logGroupIdentifiers,
        logStreamNames: logStreamNames,
        logEventFilterPattern: filterPattern
    });
    try{
        const response = await client.send(command);
        handleResponseAsync(response);
    } catch (err){
        // Pre-stream exceptions are captured here
        console.log(err);
    }
```

Arrêtez la session Live Tail après un certain temps.

```
  /* Set a timeout to close the client. This will stop the Live Tail session. */
  setTimeout(function() {
      console.log("Client timeout");
      client.destroy();
    }, 10000);
```

- Pour plus de détails sur l'API, reportez-vous StartLiveTailà la section Référence des AWS SDK pour JavaScript API.

`StartQuery`

L'exemple de code suivant montre comment utiliser`StartQuery`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
```

```
      /** @type {string} */
      const message = err.message;
      if (message.startsWith("Query's end date and time")) {
        // This error indicates that the query's start or end date occur
        // before the log group was created.
        throw new DateOutOfBoundsError(message);
      }

      throw err;
    }
  }
```

- Pour plus de détails sur l'API, reportez-vous StartQueryà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Exécution d’une requête volumineuse

L'exemple de code suivant montre comment utiliser CloudWatch Logs pour interroger plus de 10 000 enregistrements.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code AWS.

Il s’agit du point d’entrée.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
```

```
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}

const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
  logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
  dateRange: [
    new Date(Number.parseInt(process.env.QUERY_START_DATE)),
    new Date(Number.parseInt(process.env.QUERY_END_DATE)),
  ],
});

await cloudWatchQuery.run();

console.log(
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found:
 ${cloudWatchQuery.results.length}`,
);
```

Il s'agit d'une classe qui divise les requêtes en plusieurs étapes si nécessaire.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  StartQueryCommand,
  GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}

export class CloudWatchQuery {
  /**
   * Run a query for all CloudWatch Logs within a certain date range.
   * CloudWatch logs return a max of 10,000 results. This class
   * performs a binary search across all of the logs in the provided
   * date range if a query returns the maximum number of results.
   *
   * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient} client
```

```
   * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
{ limit: number } }} config
   */
  constructor(client, { logGroupNames, dateRange, queryConfig }) {
    this.client = client;
    /**
     * All log groups are queried.
     */
    this.logGroupNames = logGroupNames;

    /**
     * The inclusive date range that is queried.
     */
    this.dateRange = dateRange;

    /**
     * CloudWatch Logs never returns more than 10,000 logs.
     */
    this.limit = queryConfig?.limit ?? 10000;

    /**
     * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
     */
    this.results = [];
  }

  /**
   * Run the query.
   */
  async run() {
    this.secondsElapsed = 0;
    const start = new Date();
    this.results = await this._largeQuery(this.dateRange);
    const end = new Date();
    this.secondsElapsed = (end - start) / 1000;
    return this.results;
  }

  /**
   * Recursively query for logs.
   * @param {[Date, Date]} dateRange
   * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[][]>}
   */
  async _largeQuery(dateRange) {
```

```
    const logs = await this._query(dateRange, this.limit);

    console.log(
      `Query date range: ${dateRange
        .map((d) => d.toISOString())
        .join(" to ")}. Found ${logs.length} logs.`,
    );

    if (logs.length < this.limit) {
      return logs;
    }

    const lastLogDate = this._getLastLogDate(logs);
    const offsetLastLogDate = new Date(lastLogDate);
    offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
    const subDateRange = [offsetLastLogDate, dateRange[1]];
    const [r1, r2] = splitDateRange(subDateRange);
    const results = await Promise.all([
      this._largeQuery(r1),
      this._largeQuery(r2),
    ]);
    return [logs, ...results].flat();
  }

  /**
   * Find the most recent log in a list of logs.
   * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
   */
  _getLastLogDate(logs) {
    const timestamps = logs
      .map(
        (log) =>
          log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
      )
      .filter((t) => !!t)
      .map((t) => `${t}Z`)
      .sort();

    if (!timestamps.length) {
      throw new Error("No timestamp found in logs.");
    }

    return new Date(timestamps[timestamps.length - 1]);
  }
```

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

/**
 * Starts a query and waits for it to complete.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 */
async _query(dateRange, maxLogs) {
  try {
    const { queryId } = await this._startQuery(dateRange, maxLogs);
    const { results } = await this._waitUntilQueryDone(queryId);
    return results ?? [];
  } catch (err) {
    /**
     * This error is thrown when StartQuery returns an error indicating
     * that the query's start or end date occur before the log group was
     * created.
     */
    if (err instanceof DateOutOfBoundsError) {
      return [];
    }
    throw err;
  }
}

/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
```

```
          queryString: "fields @timestamp, @message | sort @timestamp asc",
          startTime: startDate.valueOf(),
          endTime: endDate.valueOf(),
          limit: maxLogs,
        }),
      );
    } catch (err) {
      /** @type {string} */
      const message = err.message;
      if (message.startsWith("Query's end date and time")) {
        // This error indicates that the query's start or end date occur
        // before the log group was created.
        throw new DateOutOfBoundsError(message);
      }

      throw err;
    }
  }

  /**
   * Call GetQueryResultsCommand until the query is done.
   * @param {string} queryId
   */
  _waitUntilQueryDone(queryId) {
    const getResults = async () => {
      const results = await this._getQueryResults(queryId);
      const queryDone = [
        "Complete",
        "Failed",
        "Cancelled",
        "Timeout",
        "Unknown",
      ].includes(results.status);

      return { queryDone, results };
    };

    return retry(
      { intervalInMs: 1000, maxRetries: 60, quiet: true },
      async () => {
        const { queryDone, results } = await getResults();
        if (!queryDone) {
          throw new Error("Query not done.");
        }
```

```
        return results;
      },
    );
  }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - GetQueryResults
  - StartQuery

Utilisent des événements planifiés pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par un événement EventBridge planifié par Amazon.

SDK pour JavaScript (v3)

Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

CodeBuild exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec CodeBuild.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Actions

`CreateProject`

L'exemple de code suivant montre comment utiliser`CreateProject`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un projet.

```
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;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateProjectà la section Référence des AWS SDK pour JavaScript API.

Exemples d'Amazon Cognito Identity utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Cognito Identity.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Scénarios

Scénarios

Créer une application Amazon Textract Explorer

L'exemple de code suivant montre comment explorer la sortie Amazon Textract via une application interactive.

SDK pour JavaScript (v3)

> Montre comment utiliser le AWS SDK pour JavaScript pour créer une application React qui utilise Amazon Textract pour extraire des données d'une image de document et les afficher sur une page Web interactive. Cet exemple s'exécute dans un navigateur Web et nécessite une identité

Amazon Cognito authentifiée pour les informations d'identification. Il utilise Amazon Simple Storage Service (Amazon S3) pour le stockage et, pour les notifications, il interroge une file d'attente Amazon Simple Queue Service (Amazon SQS) abonnée à une rubrique Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Exemples de fournisseurs d'identité Amazon Cognito utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du fournisseur d' AWS SDK pour JavaScript identité (v3) avec Amazon Cognito.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Actions

- Scénarios

Mise en route

Bonjour Amazon Cognito

L'exemple de code suivant montre comment faire ses premiers pas avec Amazon Cognito.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- Pour plus de détails sur l'API, reportez-vous ListUserPoolsà la section Référence des AWS SDK pour JavaScript API.

Actions

`AdminGetUser`

L'exemple de code suivant montre comment utiliser`AdminGetUser`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

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

- Pour plus de détails sur l'API, reportez-vous AdminGetUserà la section Référence des AWS SDK pour JavaScript API.

`AdminInitiateAuth`

L'exemple de code suivant montre comment utiliser`AdminInitiateAuth`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

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

- Pour plus de détails sur l'API, reportez-vous AdminInitiateAuthà la section Référence des AWS SDK pour JavaScript API.

`AdminRespondToAuthChallenge`

L'exemple de code suivant montre comment utiliser`AdminRespondToAuthChallenge`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
  username,
  totp,
  session,
}) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AdminRespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: totp,
```

```
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

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

- Pour plus de détails sur l'API, reportez-vous AdminRespondToAuthChallengeà la section Référence des AWS SDK pour JavaScript API.

`AssociateSoftwareToken`

L'exemple de code suivant montre comment utiliser`AssociateSoftwareToken`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

- Pour plus de détails sur l'API, reportez-vous AssociateSoftwareTokenà la section Référence des AWS SDK pour JavaScript API.

`ConfirmDevice`

L'exemple de code suivant montre comment utiliser`ConfirmDevice`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmDeviceCommand({
    DeviceKey: deviceKey,
    AccessToken: accessToken,
    DeviceSecretVerifierConfig: {
      PasswordVerifier: passwordVerifier,
      Salt: salt,
    },
  });

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

- Pour plus de détails sur l'API, reportez-vous ConfirmDeviceà la section Référence des AWS SDK pour JavaScript API.

`ConfirmSignUp`

L'exemple de code suivant montre comment utiliser`ConfirmSignUp`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

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

- Pour plus de détails sur l'API, reportez-vous ConfirmSignUpà la section Référence des AWS SDK pour JavaScript API.

`DeleteUser`

L'exemple de code suivant montre comment utiliser`DeleteUser`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
```

```
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteUserà la section Référence des AWS SDK pour JavaScript API.

`InitiateAuth`

L'exemple de code suivant montre comment utiliser`InitiateAuth`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
    AuthParameters: {
      USERNAME: username,
      PASSWORD: password,
    },
    ClientId: clientId,
```

```
  });

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

- Pour plus de détails sur l'API, reportez-vous InitiateAuthà la section Référence des AWS SDK pour JavaScript API.

`ListUsers`

L'exemple de code suivant montre comment utiliser`ListUsers`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

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

- Pour plus de détails sur l'API, reportez-vous ListUsersà la section Référence des AWS SDK pour JavaScript API.

`ResendConfirmationCode`

L'exemple de code suivant montre comment utiliser`ResendConfirmationCode`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

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

- Pour plus de détails sur l'API, reportez-vous ResendConfirmationCodeà la section Référence des AWS SDK pour JavaScript API.

`RespondToAuthChallenge`

L'exemple de code suivant montre comment utiliser`RespondToAuthChallenge`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
```

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

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

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

- Pour plus de détails sur l'API, reportez-vous RespondToAuthChallengeà la section Référence des AWS SDK pour JavaScript API.

`SignUp`

L'exemple de code suivant montre comment utiliser`SignUp`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
```

```
    UserAttributes: [{ Name: "email", Value: email }],
  });

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

- Pour plus de détails sur l'API, reportez-vous SignUpà la section Référence des AWS SDK pour JavaScript API.

`UpdateUserPool`

L'exemple de code suivant montre comment utiliser`UpdateUserPool`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
```

```
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Pour plus de détails sur l'API, reportez-vous UpdateUserPoolà la section Référence des AWS SDK pour JavaScript API.

`VerifySoftwareToken`

L'exemple de code suivant montre comment utiliser`VerifySoftwareToken`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
```

```
    UserCode: totp,
  });

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

- Pour plus de détails sur l'API, reportez-vous VerifySoftwareTokenà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Confirmation automatique des utilisateurs connus avec une fonction Lambda

L'exemple de code suivant illustre comment confirmer automatiquement les utilisateurs Amazon Cognito connus avec une fonction Lambda.

- Configurez un groupe d'utilisateurs pour appeler une fonction Lambda pour le déclencheur `PreSignUp`.
- Inscription d'un utilisateur avec Amazon Cognito.
- La fonction Lambda analyse une table DynamoDB et confirme automatiquement les utilisateurs connus.
- Connectez-vous en tant que nouvel utilisateur, puis nettoyez les ressources.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Configurez une exécution interactive de type « Scénario ». Les exemples JavaScript (v3) partagent un générateur de scénarios pour rationaliser les exemples complexes. Le code source complet est activé GitHub.

```
import { AutoConfirm } from "./scenario-auto-confirm.js";
```

```
/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      UserEmail: "test_email_1@example.com",
    },
    {
      UserName: "test_user_2",
      UserEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      UserEmail: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito
 authentication behavior.",
  });
}
```

Ce scénario illustre la confirmation automatique d'un utilisateur connu. Il orchestre les étapes de l'exemple.

```
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
} from "./steps-common.js";
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
```

```
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
```

```
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (/** @type {State} */ state) => {
    const [_, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
 "${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (/** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (/** @type {State} */ state) => state.users[0].UserName,
```

```
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (/** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    if (user) {
      state.errors.push(
        new Error(
          `The user "${state.selectedUser}" already exists in the user pool
 "${state.UserPoolId}".`,
        ),
      );
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
 numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);
```

```
const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (/** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({
        region: state.stackRegion,
        userPoolClientId: state.UserPoolClientId,
        username: state.selectedUser,
        email: state.users.find((u) => u.UserName === state.selectedUser)
          .UserEmail,
        password,
      });

    let [_, err] = await signUp(state.password);

    while (err?.name === "InvalidPasswordException") {
      console.warn("The password you entered was invalid.");
      await createPassword.handle(state);
      [_, err] = await signUp(state.password);
    }

    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (/** @type {State} */ state) => {
```

```
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
    if (logStreamErr) {
      state.errors.push(logStreamErr);
      return;
    }

    console.log(
      `Getting some recent events from log stream "${logStream.logStreamName}"`,
    );
    const [logEvents, logEventsErr] = await getLogEvents({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
      eventCount: 10,
      logStreamName: logStream.logStreamName,
    });
    if (logEventsErr) {
      state.errors.push(logEventsErr);
      return;
    }

    console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
```

```
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });

    if (err?.name === "PasswordResetRequiredException") {
      state.errors.push(new Error("Please reset your password."));
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with: ${state.token.slice(0,
 11)}`,
  { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [_, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
```

```
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);

const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State}*/ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
```

```
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
  );
```

Ces étapes sont partagées avec d'autres scénarios.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
```

```
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);
```

Un gestionnaire pour le déclencheur `PreSignUp` avec une fonction Lambda.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "./user-repository";
import { DynamoDBUserRepository } from "./user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user '${event.userName}'`,
    );
```

```
    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }

    const eventEmail = this.getEventUserEmail(event);
    console.log(`Looking up email ${eventEmail}.`);
    const storedUserInfo =
      await this.userRepository.getUserInfoByEmail(eventEmail);

    if (!storedUserInfo) {
      console.log(
        `Email ${eventEmail} not found. Email verification is required.`,
      );
      return event;
    }

    if (storedUserInfo.UserName !== event.userName) {
      console.log(
        `UserEmail ${eventEmail} found, but stored UserName
 '${storedUserInfo.UserName}' does not match supplied UserName '${event.userName}'.
 Verification is required.`,
      );
    } else {
      console.log(
        `UserEmail ${eventEmail} found with matching UserName
 ${storedUserInfo.UserName}. User is confirmed.`,
      );
      event.response.autoConfirmUser = true;
      event.response.autoVerifyEmail = true;
    }
    return event;
  }
}

const createPreSignUpHandler = (): PreSignUpHandler => {
  const tableName = process.env.TABLE_NAME;
  if (!tableName) {
    throw new Error("TABLE_NAME environment variable is not set");
  }

  const userRepository = new DynamoDBUserRepository(tableName);
  return new PreSignUpHandler(userRepository);
};
```

```
export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

Module d'actions de CloudWatch journalisation.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
 unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
```

```
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[] |
 null, unknown]>}
 */
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};
```

Module d’actions Amazon Cognito.

```
import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
```

```
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 *   password: string
```

```
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const response = await cognitoClient.send(
      new SignUpCommand({
        ClientId: userPoolClientId,
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 flow.
 * @param {{ region: string, clientId: string, username: string, password: string }}
 config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
```

```
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
```

```
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

Module d'actions DynamoDB.

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

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }}
 config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
 null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Inscription d'un utilisateur auprès d'un groupe d'utilisateurs nécessitant l'authentification MFA

L'exemple de code suivant illustre comment :

- Inscrivez et confirmez un utilisateur avec un nom d'utilisateur, un mot de passe et une adresse e-mail.
- Configurez l'authentification multifactorielle en associant une application MFA à l'utilisateur.
- Connectez-vous à l'aide d'un mot de passe et d'un code MFA.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour une expérience optimale, clonez le GitHub référentiel et exécutez cet exemple. Le code suivant représente un échantillon de l'exemple d'application complet.

```
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
```

```
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up'
 command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    logger.log("Signing up.");
    await signUp({ clientId, username, password, email });
    logger.log(`Signed up. A confirmation email has been sent to: ${email}.`);
    logger.log(
      `Run 'confirm-sign-up ${username} <code>' to confirm your account.`,
    );
  } catch (err) {
    logger.error(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
```

```
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-
sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
```

```
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    logger.log("Confirming user.");
    await confirmSignUp({ clientId, username, code });
    logger.log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
    );
  } catch (err) {
    logger.error(err);
  }
};

export { confirmSignUpHandler };

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

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

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../../actions/associate-software-
token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
```

```
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-
token' to finish the setup.",
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-initiate-
auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;
```

```
  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);

    logger.log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      logger.log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      logger.log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    logger.error(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

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

```
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../actions/admin-respond-to-auth-
challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-
respond-to-auth-challenge' command.`,
    );
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an
 argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
```

```
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    logger.log("Successfully authenticated.");
  } catch (err) {
    logger.error(err);
  }
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-
software-token' command.`,
    );
```

```
  }
};
const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;

  try {
    validateTotp(totp);

    logger.log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    logger.log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    logger.error(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

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

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - AdminGetUser

- AdminInitiateAuth
- AdminRespondToAuthChallenge
- AssociateSoftwareToken
- ConfirmDevice
- ConfirmSignUp
- InitiateAuth
- ListUsers
- ResendConfirmationCode
- RespondToAuthChallenge
- SignUp
- VerifySoftwareToken

Exemples d'Amazon Comprehend utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Comprehend.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Scénarios

Scénarios

Créer une application de streaming Amazon Transcribe

L'exemple de code suivant montre comment créer une application qui enregistre, transcrit et traduit de l'audio en direct en temps réel, et envoie les résultats par e-mail.

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Transcribe afin de créer une application qui enregistre, transcrit et traduit de l'audio en direct en temps réel, et envoie les résultats par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Création d'un chatbot Amazon Lex

L'exemple de code suivant montre comment créer un chatbot pour interagir avec les visiteurs de votre site Web.

SDK pour JavaScript (v3)

Indique comment utiliser l'API Amazon Lex pour créer un chatbot dans une application Web afin d'interagir avec les visiteurs de votre site Web.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet de création d'un chatbot Amazon Lex dans le guide du AWS SDK pour JavaScript développeur.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Créez une application pour analyser les commentaires des clients

L'exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour JavaScript (v3)

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub. Les extraits suivants montrent comment le AWS SDK pour JavaScript est utilisé dans les fonctions Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });
```

```
  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });
```

```
  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
```

```
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly

- Amazon Textract
- Amazon Translate

Exemples d'Amazon DocumentDB utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon DocumentDB.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Exemples sans serveur

Invocation d'une fonction Lambda à partir d'un déclencheur Amazon DocumentDB

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux de modifications DocumentDB. La fonction récupère les données utiles DocumentDB et journalise le contenu de l'enregistrement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de. 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));
};
```

Utilisation d'un événement Amazon DocumentDB avec Lambda à l'aide de 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));
};
```

Exemples DynamoDB utilisant le SDK JavaScript pour (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec DynamoDB.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Hello DynamoDB

L'exemple de code suivant montre comment démarrer avec DynamoDB.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Pour plus de détails sur l'utilisation de DynamoDB AWS SDK pour JavaScript dans, consultez la section Programmation de DynamoDB avec. JavaScript

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

const client = new DynamoDBClient({});

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

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez une table pouvant contenir des données vidéo.
- Insérer, récupérez et mettez à jour un seul film dans la table.
- Écrivez des données vidéo dans la table à partir d'un exemple de fichier JSON.
- Recherchez les films sortis au cours d'une année donnée.
- Recherchez les films sortis au cours d'une plage d'années spécifique.
- Supprimez un film de la table, puis supprimez la table.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { readFileSync } from "node:fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

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

export const main = async () => {
  /**
   * Create a table.
   */
```

```
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });

  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

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

  /**
```

```
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
 1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);
```

```
  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
   */

  log("Deleting a single movie from the table.");
  const deleteCommand = new DeleteCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
  });
  await docClient.send(deleteCommand);
  log("Movie deleted.");

  /**
   * Upload a batch of movies.
   */

  log("Adding movies from local JSON file.");
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );
  const movies = JSON.parse(file.toString());
```

```
  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        [tableName]: putRequests,
      },
    });

    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
```

```
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression. Scan
 will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
```

```
   */

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

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

Actions

`BatchExecuteStatement`

L'exemple de code suivant montre comment utiliser`BatchExecuteStatement`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un lot d'éléments à l'aide de PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });

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

Obtenez un lot d'éléments à l'aide de PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
```

```
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

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

Mettez à jour un lot d'éléments à l'aide de PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

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

```
};
```

Supprimez un lot d'éléments à l'aide de PartiQL.

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

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

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

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour JavaScript API.

`BatchGetItem`

L'exemple de code suivant montre comment utiliser`BatchGetItem`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir 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;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS SDK pour JavaScript API.

`BatchWriteItem`

L'exemple de code suivant montre comment utiliser`BatchWriteItem`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir BatchWrite.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node:fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

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

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

```
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year' is
 recommended
        // to account for duplicate titles.
        BatchWriteMoviesTable: putRequests,
      },
    });

    await docClient.send(command);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour JavaScript API.

`CreateTable`

L'exemple de code suivant montre comment utiliser`CreateTable`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const client = new DynamoDBClient({});

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

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

- Pour plus d’informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour JavaScript API.

`DeleteItem`

L'exemple de code suivant montre comment utiliser`DeleteItem`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir 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;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour JavaScript API.

`DeleteTable`

L'exemple de code suivant montre comment utiliser`DeleteTable`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

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

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour JavaScript API.

`DescribeTable`

L'exemple de code suivant montre comment utiliser`DescribeTable`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK pour JavaScript API.

`DescribeTimeToLive`

L'exemple de code suivant montre comment utiliser`DescribeTimeToLive`.

SDK pour JavaScript (v3)

Décrivez la configuration Durée de vie (TTL) sur une table DynamoDB existante à l'aide du kit AWS SDK pour JavaScript.

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

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

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

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

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

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- Pour plus de détails sur l'API, reportez-vous DescribeTimeToLiveà la section Référence des AWS SDK pour JavaScript API.

`ExecuteStatement`

L'exemple de code suivant montre comment utiliser`ExecuteStatement`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un élément à l'aide de PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

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

```
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

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

Obtenez un élément à l'aide de PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

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

Mettez à jour d'un élément à l'aide de PartiQL.

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

```
import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

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

Supprimez un élément à l'aide de PartiQL.

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

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

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

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

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

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour JavaScript API.

`GetItem`

L'exemple de code suivant montre comment utiliser`GetItem`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir 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;
};
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour JavaScript API.

`ListTables`

L'exemple de code suivant montre comment utiliser`ListTables`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const client = new DynamoDBClient({});

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

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

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour JavaScript API.

`PutItem`

L'exemple de code suivant montre comment utiliser`PutItem`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir 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;
};
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour JavaScript API.

`Query`

L'exemple de code suivant montre comment utiliser`Query`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir 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;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

`Scan`

L'exemple de code suivant montre comment utiliser`Scan`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir 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;
};
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK pour JavaScript .

`UpdateItem`

L'exemple de code suivant montre comment utiliser`UpdateItem`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple utilise le client de document pour simplifier l'utilisation d'éléments dans DynamoDB. Pour plus de détails sur l'API, voir 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;
};
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

`UpdateTimeToLive`

L'exemple de code suivant montre comment utiliser`UpdateTimeToLive`.

SDK pour JavaScript (v3)

Activation de la TTL (Durée de vie) sur une table DynamoDB existante.

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

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') => {

    const client = new DynamoDBClient({
        region: region,
```

```
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

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

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

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Désactivation de la TTL (Durée de vie) sur une table DynamoDB existante.

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

export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1') => {

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

    const params = {
        TableName: tableName,
```

```
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

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

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Pour plus de détails sur l'API, reportez-vous UpdateTimeToLiveà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer une application pour soumettre des données à une table DynamoDB

L'exemple de code suivant montre comment créer une application qui soumet des données à une table Amazon DynamoDB et vous avertit lorsqu'un utilisateur met à jour la table.

SDK pour JavaScript (v3)

> Cet exemple montre comment créer une application qui permet aux utilisateurs de soumettre des données à une table Amazon DynamoDB et d'envoyer un message texte à l'administrateur à l'aide d'Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SNS

Comparaison de plusieurs valeurs avec un seul attribut

L'exemple de code suivant montre comment comparer plusieurs valeurs avec un seul attribut dans DynamoDB.

- Utilisez l'opérateur IN pour comparer plusieurs valeurs avec un seul attribut.
- Comparez l'opérateur IN avec plusieurs conditions OR.
- Comprenez les avantages liés à l'utilisation d'IN en matière de performances et de complexité d'expression.

SDK pour JavaScript (v3)

Comparez plusieurs valeurs avec un seul attribut en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  ScanCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Query or scan a DynamoDB table to find items where an attribute matches any value
 from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 attribute
 * against multiple possible values, which is more efficient than using multiple OR
 conditions.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against the
 values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key attribute
 for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the matching
 items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the filter expression using the IN operator
  const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
 `:val${index}`).join(', ')})`;

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
```

```
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
```

```
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against the
 values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key attribute
 for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the matching
 items
 */
async function compareWithOrConditions(
```

```
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }

  // Create the filter expression using multiple OR conditions
  const filterConditions = valuesList.map((_, index) => `${attributeName} = :val
${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
```

```
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }
```

```
      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}
```

Exemple d'utilisation de la comparaison de plusieurs valeurs avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const attributeName = "Category";
  const valuesList = ["Electronics", "Computers", "Accessories"];

  console.log(`Searching for products in any of these categories:
 ${valuesList.join(', ')}`);

  try {
    // Using the IN operator (recommended approach)
    console.log("\nApproach 1: Using the IN operator");
    const response = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList
```

```
    );

    console.log(`Found ${response.Count} products in the specified categories`);

    // Using multiple OR conditions (alternative approach)
    console.log("\nApproach 2: Using multiple OR conditions");
    const response2 = await compareWithOrConditions(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response2.Count} products in the specified categories`);

    // Example with a query operation
    console.log("\nQuerying a specific manufacturer's products in multiple
categories");
    const partitionKeyName = "Manufacturer";
    const partitionKeyValue = "Acme";

    const response3 = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList,
      partitionKeyName,
      partitionKeyValue
    );

    console.log(`Found ${response3.Count} Acme products in the specified
categories`);

    // Explain the benefits of using the IN operator
    console.log("\nBenefits of using the IN operator:");
    console.log("1. More concise expression compared to multiple OR conditions");
    console.log("2. Better readability and maintainability");
    console.log("3. Potentially better performance with large value lists");
    console.log("4. Simpler code that's less prone to errors");
    console.log("5. Easier to modify when adding or removing values");

  } catch (error) {
    console.error("Error:", error);
  }
```

```
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript .
  - Interrogation
  - Analyser

Mise à jour sous certaines conditions du TTL d'un élément

L'exemple de code suivant montre comment mettre à jour la TTL d'un élément de manière conditionnelle.

SDK pour JavaScript (v3)

Mettez à jour de la TTL sur un élément DynamoDB existant dans une table, avec une condition.

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

export const updateItemConditional = async (tableName, partitionKey, sortKey, region
 = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

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

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

```
    };

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

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-sort-
key-value');
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Comptage des opérateurs d’expression

L’exemple de code suivant montre comment compter les opérateurs d’expression dans DynamoDB.

- Comprenez la limite de 300 opérateurs de DynamoDB.
- Comptage des opérateurs dans les expressions complexes.
- Optimisation des expressions pour respecter les limites.

SDK pour JavaScript (v3)

Démontrez le comptage des opérateurs d’expression à l’aide du kit AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
```

```
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 conditions
 * @returns {Object} - Object containing the filter expression and attribute values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};

  // Generate the specified number of conditions
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        condition = `attribute${i} < ${valueKey}`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
      case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
      case 4:
        condition = `attribute_exists(attribute${i})`;
```

```
        break;
    }

    conditions.push(condition);
  }

  // Join the conditions with AND or OR
  const operator = useAnd ? " AND " : " OR ";
  const filterExpression = conditions.join(operator);

  // Calculate the operator count
  // Each condition has 1 operator (=, >, <, contains, attribute_exists)
  // Each AND or OR between conditions is 1 operator
  const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount -
 1 : 0);

  return {
    filterExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute values
 */
function createComplexUpdateExpression(operationsCount) {
  // Initialize the expression parts and attribute values
  const setOperations = [];
  const expressionAttributeValues = {};

  // Generate the specified number of SET operations
  for (let i = 0; i < operationsCount; i++) {
    // Alternate between different types of SET operations
    let operation;
    const valueKey = `:val${i}`;

    switch (i % 3) {
```

```
      case 0:
        // Simple assignment (1 operator: =)
        operation = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        // Addition (2 operators: = and +)
        operation = `attribute${i} = attribute${i} + ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        // Conditional assignment with if_not_exists (2 operators: = and
 if_not_exists)
        operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
    }

    setOperations.push(operation);
  }

  // Create the update expression
  const updateExpression = `SET ${setOperations.join(", ")}`;

  // Calculate the operator count
  // Each operation has 1-2 operators as noted above
  let operatorCount = 0;
  for (let i = 0; i < operationsCount; i++) {
    operatorCount += (i % 3 === 0) ? 1 : 2;
  }

  return {
    updateExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
async function testOperatorLimit(
  config,
  tableName,
  key,
  operatorCount
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create a complex update expression with the specified operator count
  const { updateExpression, expressionAttributeValues, operatorCount: actualCount }
 =
    createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to get
 close to target count

  console.log(`Generated update expression with approximately ${actualCount}
 operators`);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Attempt the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      message: `Operation succeeded with ${actualCount} operators`,
      data: response
    };
  } catch (error) {
    // Check if the error is due to exceeding the operator limit
```

```
    if (error.name === "ValidationException" &&
        error.message.includes("too many operators")) {
      return {
        success: false,
        message: `Operation failed: ${error.message}`,
        operatorCount: actualCount
      };
    }

    // Return other errors
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      error
    };
  }
}

/**
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
  config,
  tableName,
  key,
  totalOperations
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Calculate how many operations we can safely include in each batch
  // Using 150 as a conservative limit (well below 300)
  const operationsPerBatch = 100;
  const batchCount = Math.ceil(totalOperations / operationsPerBatch);
```

```
 console.log(`Breaking down ${totalOperations} operations into ${batchCount}
batches`);

 const results = [];

 // Process each batch
 for (let batch = 0; batch < batchCount; batch++) {
   // Calculate the operations for this batch
   const batchStart = batch * operationsPerBatch;
   const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
   const batchSize = batchEnd - batchStart;

   console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
operations`);

   // Create an update expression for this batch
   const { updateExpression, expressionAttributeValues, operatorCount } =
     createComplexUpdateExpression(batchSize);

   // Define the update parameters
   const params = {
     TableName: tableName,
     Key: key,
     UpdateExpression: updateExpression,
     ExpressionAttributeValues: expressionAttributeValues,
     ReturnValues: "UPDATED_NEW"
   };

   try {
     // Perform the update operation for this batch
     const response = await docClient.send(new UpdateCommand(params));

     results.push({
       batch: batch + 1,
       success: true,
       operatorCount,
       attributes: response.Attributes
     });
   } catch (error) {
     results.push({
       batch: batch + 1,
       success: false,
       operatorCount,
```

```
        error: error.message
      });

      // Stop processing if an error occurs
      break;
    }
  }

  return {
    totalBatches: batchCount,
    results
  };
}

/**
 * Count operators in a DynamoDB expression based on the rules in the documentation.
 *
 * This function demonstrates how operators are counted according to the
 * DynamoDB documentation.
 *
 * @param {string} expression - The DynamoDB expression to analyze
 * @returns {Object} - Breakdown of operator counts
 */
function countOperatorsInExpression(expression) {
  // Initialize counters for different operator types
  const counts = {
    comparisonOperators: 0,
    logicalOperators: 0,
    functions: 0,
    arithmeticOperators: 0,
    specialOperators: 0,
    total: 0
  };

  // Count comparison operators (=, <>, <, <=, >, >=)
  const comparisonRegex = /[^<>]=[^=]|<>|<=|>=|[^<]>[^=]|[^>]<[^=]/g;
  const comparisonMatches = expression.match(comparisonRegex) || [];
  counts.comparisonOperators = comparisonMatches.length;

  // Count logical operators (AND, OR, NOT)
  const andMatches = expression.match(/\bAND\b/g) || [];
  const orMatches = expression.match(/\bOR\b/g) || [];
  const notMatches = expression.match(/\bNOT\b/g) || [];
```

```
  counts.logicalOperators = andMatches.length + orMatches.length +
 notMatches.length;

  // Count functions (attribute_exists, attribute_not_exists, attribute_type,
 begins_with, contains, size)
  const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
  const functionMatches = expression.match(functionRegex) || [];
  counts.functions = functionMatches.length;

  // Count arithmetic operators (+ and -)
  const arithmeticMatches = expression.match(/[a-zA-Z0-9_)\]]\s*[\+\-]\s*[a-zA-
Z0-9_(:]/g) || [];
  counts.arithmeticOperators = arithmeticMatches.length;

  // Count special operators (BETWEEN, IN)
  const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
  const inMatches = expression.match(/\bIN\b/g) || [];
  counts.specialOperators = betweenMatches.length + inMatches.length;

  // Add extra operators for BETWEEN (each BETWEEN includes an AND)
  counts.logicalOperators += betweenMatches.length;

  // Calculate total
  counts.total = counts.comparisonOperators +
                 counts.logicalOperators +
                 counts.functions +
                 counts.arithmeticOperators +
                 counts.specialOperators;

  return counts;
}
```

Exemple d'utilisation de l'opérateur d'expression comptant avec AWS SDK pour JavaScript.

```
/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
```

```
 const key = { ProductId: "P12345" };

 console.log("Demonstrating DynamoDB expression operator counting and the 300
operator limit");

 try {
   // Example 1: Analyze a simple expression
   console.log("\nExample 1: Analyzing a simple expression");
   const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
(:cat1, :cat2, :cat3)";
   const simpleCount = countOperatorsInExpression(simpleExpression);

   console.log(`Expression: ${simpleExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
   console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
   console.log(`- Functions: ${simpleCount.functions}`);
   console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
   console.log(`- Special operators: ${simpleCount.specialOperators}`);
   console.log(`- Total operators: ${simpleCount.total}`);

   // Example 2: Analyze a complex expression
   console.log("\nExample 2: Analyzing a complex expression");
   const complexExpression =
     "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
     "(Price > :price AND contains(Description, :keyword) AND " +
     "(Rating >= :minRating OR Reviews > :minReviews))";
   const complexCount = countOperatorsInExpression(complexExpression);

   console.log(`Expression: ${complexExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
   console.log(`- Logical operators: ${complexCount.logicalOperators}`);
   console.log(`- Functions: ${complexCount.functions}`);
   console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
   console.log(`- Special operators: ${complexCount.specialOperators}`);
   console.log(`- Total operators: ${complexCount.total}`);

   // Example 3: Test approaching the operator limit
   console.log("\nExample 3: Testing an expression approaching the operator
limit");
   const approachingLimit = await testOperatorLimit(config, tableName, key, 290);
   console.log(approachingLimit.message);
```

```
    // Example 4: Test exceeding the operator limit
    console.log("\nExample 4: Testing an expression exceeding the operator limit");
    const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
    console.log(exceedingLimit.message);

    // Example 5: Breaking down a complex expression
    console.log("\nExample 5: Breaking down a complex expression into multiple
 operations");
    const breakdownResult = await breakDownComplexExpression(config, tableName, key,
 500);
    console.log(`Processed ${breakdownResult.results.length} of
 ${breakdownResult.totalBatches} batches`);

    // Explain the operator counting rules
    console.log("\nKey points about DynamoDB expression operator counting:");
    console.log("1. The maximum number of operators in any expression is 300");
    console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
 operator");
    console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
    console.log("4. Each function call (attribute_exists, contains, etc.) counts as
 1 operator");
    console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
    console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND within
 it)");
    console.log("7. IN counts as 1 operator regardless of the number of values");
    console.log("8. Parentheses for grouping and attribute paths don't count as
 operators");
    console.log("9. When you exceed the limit, the error always reports '301
 operators'");
    console.log("10. For complex operations, break them into multiple smaller
 operations");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Création d’une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour JavaScript (v3)

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l’aide d’Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l’origine de cet exemple, consultez l’article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Création d’une table avec le débit à chaud activé

L’exemple de code suivant montre comment créer une table avec le débit à chaud activé.

SDK pour JavaScript (v3)

Créez une table DynamoDB avec le paramètre de débit à chaud à l’aide du kit AWS SDK pour JavaScript.

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

export async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
```

```
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
```

```
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
    const response = await ddbClient.send(command);
    console.log(response);
    return response;
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
  'example-table',
  'pk',
  'sk',
  'gsiKey',
  'data',
  10, 10, 5, 5,
  'example-index',
  5, 5, 2, 2
);
*/
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour JavaScript API.

Création d’un élément avec un TTL

L’exemple de code suivant montre comment créer un élément avec une TTL.

SDK pour JavaScript (v3)

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

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

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

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

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

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

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

```
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour JavaScript API.

Suppression de données à l’aide de PartiQL DELETE

L’exemple de code suivant montre comment supprimer des données à l’aide des instructions PartiQL DELETE.

SDK pour JavaScript (v3)

Supprimez des éléments d'une table DynamoDB à l'aide des instructions PARTIQL DELETE avec. AWS SDK pour JavaScript

```
/**
 * This example demonstrates how to delete items from a DynamoDB table using
 PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByPartitionKey = async (
  tableName: string,
```

```
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
```

```
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item with a condition to ensure the delete only happens if a condition
 is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${conditionAttribute} = ?`,
    Parameters: [partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted with condition successfully");
    return data;
```

```
  } catch (err) {
    console.error("Error deleting item with condition:", err);
    throw err;
  }
};

/**
 * Batch delete multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param keys - Array of objects containing key information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchDeleteItems = async (
  tableName: string,
  keys: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    sortKeyName?: string;
    sortKeyValue?: string | number;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each delete
  const statements = keys.map((key) => {
    if (key.sortKeyName && key.sortKeyValue !== undefined) {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ? AND
 ${key.sortKeyName} = ?`,
        Parameters: [key.partitionKeyValue, key.sortKeyValue],
      };
    } else {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?`,
        Parameters: [key.partitionKeyValue],
      };
    }
  });

  const params = {
    Statements: statements,
  };
```

```
  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch deleted successfully");
    return data;
  } catch (err) {
    console.error("Error batch deleting items:", err);
    throw err;
  }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemsByFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items deleted by filter successfully");
    return data;
  } catch (err) {
    console.error("Error deleting items by filter:", err);
    throw err;
  }
};

/**
```

```
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {
  // Delete an item by partition key (simple primary key)
  await deleteItemByPartitionKey("UsersTable", "userId", "user123");

  // Delete an item by composite key (partition key + sort key)
  await deleteItemByCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789"
  );

  // Delete with a condition
  await deleteItemWithCondition(
    "UsersTable",
    "userId",
    "user789",
    "userStatus",
    "inactive"
  );

  // Batch delete multiple items
  await batchDeleteItems("UsersTable", [
    { partitionKeyName: "userId", partitionKeyValue: "user234" },
    { partitionKeyName: "userId", partitionKeyValue: "user345" },
  ]);

  // Batch delete items with composite keys
  await batchDeleteItems("OrdersTable", [
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order567",
      sortKeyName: "productId",
      sortKeyValue: "prod123",
    },
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order678",
      sortKeyName: "productId",
      sortKeyValue: "prod456",
    },
```

```
  ]);

  // Delete items by filter (use with caution)
  await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Insertion de données à l'aide de PartiQL INSERT

L'exemple de code suivant montre comment insérer des données à l'aide des instructions PartiQL INSERT.

SDK pour JavaScript (v3)

Insérez des éléments dans une table DynamoDB à l'aide des instructions PartiQL INSERT avec. AWS SDK pour JavaScript

```
/**
 * This example demonstrates how to insert items into a DynamoDB table using
 PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
 */
```

```
export const insertItem = async (tableName: string, item: Record<string, any>) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Convert the item to a string representation for PartiQL
  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item:", err);
    throw err;
  }
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string,
 any>[]) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
    return {
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
    Statements: statements,
```

```
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
    return data;
  } catch (err) {
    console.error("Error batch inserting items:", err);
    throw err;
  }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
  tableName: string,
  item: Record<string, any>,
  partitionKeyName: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
  const partitionKeyValue = JSON.stringify(item[partitionKeyName]);

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
 attribute_not_exists(${partitionKeyName})`,
    Parameters: [{ S: partitionKeyValue }],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item with condition:", err);
```

```
    throw err;
  }
};

/**
 * Example usage showing how to insert items with different index types
 */
export const insertExamples = async () => {
  // Example table with a simple primary key (just partition key)
  const simpleKeyItem = {
    userId: "user123",
    name: "John Doe",
    email: "john@example.com",
  };
  await insertItem("UsersTable", simpleKeyItem);

  // Example table with composite key (partition key + sort key)
  const compositeKeyItem = {
    orderId: "order456",
    productId: "prod789",
    quantity: 2,
    price: 29.99,
  };
  await insertItem("OrdersTable", compositeKeyItem);

  // Example with Global Secondary Index (GSI)
  // The GSI might be on the email attribute
  const gsiItem = {
    userId: "user789",
    email: "jane@example.com",
    name: "Jane Smith",
    userType: "premium",  // This could be part of a GSI
  };
  await insertItem("UsersTable", gsiItem);

  // Example with Local Secondary Index (LSI)
  // LSI uses the same partition key but different sort key
  const lsiItem = {
    orderId: "order567",  // Partition key
    productId: "prod123", // Sort key for the table
    orderDate: "2023-11-15", // Potential sort key for an LSI
    quantity: 1,
    price: 19.99,
  };
```

```
  await insertItem("OrdersTable", lsiItem);

  // Batch insert example with multiple items
  const batchItems = [
    {
      userId: "user234",
      name: "Alice Johnson",
      email: "alice@example.com",
    },
    {
      userId: "user345",
      name: "Bob Williams",
      email: "bob@example.com",
    },
  ];
  await batchInsertItems("UsersTable", batchItems);
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Invocation d'une fonction Lambda à partir d'un navigateur

L'exemple de code suivant montre comment invoquer une AWS Lambda fonction depuis un navigateur.

SDK pour JavaScript (v3)

Vous pouvez créer une application basée sur un navigateur qui utilise une AWS Lambda fonction pour mettre à jour une table Amazon DynamoDB avec les sélections des utilisateurs. Cette application utilise la AWS SDK pour JavaScript version 3.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB

- Lambda

Exécution d'opérations de requête avancées

L'exemple de code suivant montre comment exécuter des opérations de requête avancées dans DynamoDB.

- Interrogez des tables à l'aide de diverses techniques de filtrage et de conditions.
- Mettez en œuvre la pagination pour les grands ensembles de résultats.
- Utilisation d'index secondaires globaux pour les autres modèles d'accès.
- Appliquez des contrôles de cohérence en fonction des exigences de l'application.

SDK pour JavaScript (v3)

Requête avec des lectures très cohérentes utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
```

```
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

Requête à l'aide d'un index secondaire global avec AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
```

```
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
```

```
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

Requête avec pagination à l'aide AWS SDK pour JavaScript de.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
```

```
 try {
   // Create DynamoDB client
   const client = new DynamoDBClient(config);

   // Initialize variables for pagination
   let lastEvaluatedKey = undefined;
   const allItems = [];
   let pageCount = 0;

   // Loop until all pages are retrieved
   do {
     // Construct the query input
     const input = {
       TableName: tableName,
       KeyConditionExpression: "#pk = :pkValue",
       Limit: pageSize,
       ExpressionAttributeNames: {
         "#pk": partitionKeyName
       },
       ExpressionAttributeValues: {
         ":pkValue": { S: partitionKeyValue }
       }
     };

     // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous query
     if (lastEvaluatedKey) {
       input.ExclusiveStartKey = lastEvaluatedKey;
     }

     // Execute the query
     const command = new QueryCommand(input);
     const response = await client.send(command);

     // Process the current page of results
     pageCount++;
     console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

     // Add the items from this page to our collection
     if (response.Items && response.Items.length > 0) {
       allItems.push(...response.Items);
     }

     // Get the LastEvaluatedKey for the next page
```

```
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in ${pageCount}
 pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

Requête avec des filtres complexes à l'aide de AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
```

```
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
```

```
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

Requête avec une expression de filtre construite dynamiquement à l'aide de AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
```

```
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Exécution d'opérations de liste

L'exemple de code suivant montre comment exécuter des opérations de liste dans DynamoDB.

- Ajout d'éléments à un attribut de liste.
- Supprimez des éléments d'un attribut de liste.
- Mettez à jour des éléments spécifiques dans une liste par index.
- Utilisez des fonctions d'ajout de liste et d'index de liste.

SDK pour JavaScript (v3)

Démontrez les opérations de liste en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
```

```
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} =
 list_append(if_not_exists(${listName}, :empty_list), :values)`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function
```

```
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = list_append(:values,
 if_not_exists(${listName}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${index}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
  config,
  tableName,
  key,
  listName,
  index
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using REMOVE with index notation
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `REMOVE ${listName}[${index}]`,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Concatenate two lists.
 *
 * This function demonstrates how to concatenate two lists using the list_append
 function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${resultListName} =
 list_append(if_not_exists(${listName1}, :empty_list),
 if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a nested list structure.
 *
 * This function demonstrates how to create and work with nested lists.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} nestedLists - An array of arrays to create a nested list structure
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createNestedList(
  config,
  tableName,
  key,
  listName,
  nestedLists
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters to create a nested list
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = :nested_lists`,
    ExpressionAttributeValues: {
```

```
      ":nested_lists": nestedLists
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an element in a nested list.
 *
 * This function demonstrates how to update an element in a nested list
 * using multiple index notations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} outerIndex - The index in the outer list
 * @param {number} innerIndex - The index in the inner list
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedListElement(
  config,
  tableName,
  key,
  listName,
  outerIndex,
  innerIndex,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using multiple index notations
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
```

```
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemple d'utilisation d'opérations de liste avec AWS SDK pour JavaScript.

```
/**
 * Example of how to work with lists in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

  console.log("Demonstrating list operations in DynamoDB");

  try {
    // Example 1: Append elements to a list
    console.log("\nExample 1: Appending elements to a list");
    const response1 = await appendToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["laptop", "headphones", "monitor"]
    );

    console.log("Appended to list:", response1.Attributes);

    // Example 2: Prepend elements to a list
    console.log("\nExample 2: Prepending elements to a list");
    const response2 = await prependToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["keyboard", "mouse"]
    );

    console.log("Prepended to list:", response2.Attributes);

    // Get the current state of the item
    let currentItem = await getItem(config, tableName, key);
    console.log("\nCurrent state of RecentSearches:", currentItem?.RecentSearches);
```

```
// Example 3: Update a specific element in a list
console.log("\nExample 3: Updating a specific element in a list");
const response3 = await updateListElement(
  config,
  tableName,
  key,
  "RecentSearches",
  0, // Update the first element
  "mechanical keyboard" // New value
);

console.log("Updated list element:", response3.Attributes);

// Example 4: Remove an element from a list
console.log("\nExample 4: Removing an element from a list");
const response4 = await removeListElement(
  config,
  tableName,
  key,
  "RecentSearches",
  2 // Remove the third element
);

console.log("List after removing element:", response4.Attributes);

// Example 5: Create and concatenate lists
console.log("\nExample 5: Creating and concatenating lists");

// First, create two separate lists
await updateWithMultipleActions(
  config,
  tableName,
  key,
  "SET WishList = :wishlist, SavedItems = :saveditems",
  null,
  {
    ":wishlist": ["gaming laptop", "wireless earbuds"],
    ":saveditems": ["smartphone", "tablet"]
  }
);

// Then, concatenate them
const response5 = await concatenateLists(
```

```
    config,
    tableName,
    key,
    "WishList",
    "SavedItems",
    "AllItems"
  );

  console.log("Concatenated lists:", response5.Attributes);

  // Example 6: Create a nested list structure
  console.log("\nExample 6: Creating a nested list structure");
  const response6 = await createNestedList(
    config,
    tableName,
    key,
    "Categories",
    [
      ["Electronics", "Computers", "Accessories"],
      ["Books", "Magazines", "E-books"],
      ["Clothing", "Shoes", "Watches"]
    ]
  );

  console.log("Created nested list:", response6.Attributes);

  // Example 7: Update an element in a nested list
  console.log("\nExample 7: Updating an element in a nested list");
  const response7 = await updateNestedListElement(
    config,
    tableName,
    key,
    "Categories",
    0, // First inner list
    1, // Second element in that list
    "Laptops" // New value
  );

  console.log("Updated nested list element:", response7.Attributes);

  // Get the final state of the item
  currentItem = await getItem(config, tableName, key);
  console.log("\nFinal state of the item:", JSON.stringify(currentItem, null, 2));
```

```
    // Explain list operations
    console.log("\nKey points about list operations in DynamoDB:");
    console.log("1. Use list_append to add elements to a list");
    console.log("2. To append elements, use list_append(existingList,
 newElements)");
    console.log("3. To prepend elements, use list_append(newElements,
 existingList)");
    console.log("4. Use if_not_exists to handle cases where the list might not exist
 yet");
    console.log("5. Use index notation (list[0]) to access or update specific
 elements");
    console.log("6. Use REMOVE with index notation to remove elements from a list");
    console.log("7. Lists can contain elements of different types");
    console.log("8. Lists can be nested (lists of lists)");
    console.log("9. Use multiple index notations (list[0][1]) to access nested list
 elements");

  } catch (error) {
    console.error("Error:", error);
  }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
```

```
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Exécution d’opérations de mappage

L’exemple de code suivant montre comment exécuter des opérations de mappage dans DynamoDB.

- Ajoutez et mettez à jour des attributs imbriqués dans les structures de mappage.
- Suppression des champs spécifiques des mappages.
- Utilisation d’attributs de mappage profondément imbriqués.

SDK pour JavaScript (v3)

Démontrez les opérations cartographiques en utilisant AWS SDK pour JavaScript.

```
/**
 * Example of updating map attributes in DynamoDB.
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update scenarios.
 */

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeSafe(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
```

```
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {
    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update expression
 is invalid")) {

      // Create the map with the specified key-value pair
      const createParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${mapName} = :map`,
        ExpressionAttributeValues: {
          ":map": { [mapKey]: value }
        },
        ReturnValues: "UPDATED_NEW"
      };

      return await docClient.send(new UpdateCommand(createParams));
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeWithIfNotExists(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
 ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":emptyMap": {},
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.
```

```
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} path - The path to the nested attribute as an array of keys
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
      updateExpression += ` ${parentPath} = if_not_exists(${parentPath}, :emptyMap
${i}),`;
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[":value"] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 *
 * This function demonstrates how to update multiple fields in a map
 * in a single DynamoDB operation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {Object} updates - Object containing key-value pairs to update
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
  updates
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
```

```
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
```

```
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}

/**
 * Example of how to use the map attribute update functions.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating different approaches to update map attributes in
 DynamoDB");

  try {
    // Example 1: Update a map attribute that might not exist (two-step approach)
    console.log("\nExample 1: Updating a map attribute that might not exist (two-
step approach)");
    const response1 = await updateMapAttributeSafe(
      config,
      tableName,
      key,
      "Preferences",
      "Theme",
      "Dark"
    );

    console.log("Updated preferences:", response1.Attributes);

    // Example 2: Update a map attribute using if_not_exists (elegant approach)
    console.log("\nExample 2: Updating a map attribute using if_not_exists (elegant
 approach)");
    const response2 = await updateMapAttributeWithIfNotExists(
      config,
      tableName,
      key,
      "Settings",
      "NotificationsEnabled",
      true
    );
```

```
  console.log("Updated settings:", response2.Attributes);

  // Example 3: Update a deeply nested attribute
  console.log("\nExample 3: Updating a deeply nested attribute");
  const response3 = await addToNestedMap(
    config,
    tableName,
    key,
    ["Profile", "Address", "City"],
    "Seattle"
  );

  console.log("Updated nested attribute:", response3.Attributes);

  // Example 4: Update multiple fields in a map
  console.log("\nExample 4: Updating multiple fields in a map");
  const response4 = await updateMultipleMapFields(
    config,
    tableName,
    key,
    "ContactInfo",
    {
      Email: "user@example.com",
      Phone: "555-123-4567",
      PreferredContact: "Email"
    }
  );

  console.log("Updated multiple fields:", response4.Attributes);

  // Get the final state of the item
  console.log("\nFinal state of the item:");
  const item = await getItem(config, tableName, key);
  console.log(JSON.stringify(item, null, 2));

  // Explain the benefits of different approaches
  console.log("\nKey points about updating map attributes:");
  console.log("1. Use if_not_exists to handle maps that might not exist");
  console.log("2. Multiple updates can be combined in a single operation");
  console.log("3. Deeply nested attributes require creating parent maps");
  console.log("4. DynamoDB expressions are atomic - the entire update succeeds or
fails");
```

```
    console.log("5. Using a single operation is more efficient than multiple
 separate updates");

  } catch (error) {
    console.error("Error:", error);
  }
}

// Export the functions
module.exports = {
  updateMapAttributeSafe,
  updateMapAttributeWithIfNotExists,
  addToNestedMap,
  updateMultipleMapFields,
  getItem,
  exampleUsage
};

// Run the example if this file is executed directly
if (require.main === module) {
  exampleUsage();
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Exécution d’opérations d’ensemble

L’exemple de code suivant montre comment exécuter des opérations d’ensemble dans DynamoDB.

- Ajoutez des éléments à un attribut d’ensemble.
- Supprimez des éléments dans un attribut d’ensemble.
- Utilisez des opérations ADD et DELETE avec des ensembles.

SDK pour JavaScript (v3)

Démontrez les opérations du set en utilisant AWS SDK pour JavaScript.

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

```
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using ADD
  const params = {
```

```
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove elements from a set attribute.
 *
 * This function demonstrates using the DELETE operation to remove elements from a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to remove from the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
```

```
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using DELETE
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `DELETE ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
  config,
  tableName,
  key,
  setName,
```

```
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using SET
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${setName} = :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
```

```
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The new values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function replaceSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}

/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
```

```
    return {
      success: false,
      message: "Set doesn't exist or is already empty",
      item: currentItem
    };
  }

  // Get the set values
  const setValues = Array.from(currentItem[setName]);

  // If there's only one element left, remove the attribute entirely
  if (setValues.length === 1) {
    // Define the update parameters to remove the attribute
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `REMOVE ${setName}`,
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Last element removed, attribute has been deleted",
      removedValue: setValues[0]
    };
  } else {
    // Otherwise, remove just the last element
    // Create a set with just the last element
    const lastElement = setValues[setValues.length - 1];
    const setType = typeof lastElement === 'number' ? 'number' : 'string';

    // Remove the last element
    const response = await removeFromSet(
      config,
      tableName,
      key,
      setName,
      [lastElement],
      setType
    );
```

```
    return {
      success: true,
      message: "Last element removed, set still contains elements",
      removedValue: lastElement,
      remainingSet: response.Attributes[setName]
    };
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemple d'utilisation d'opérations définies avec AWS SDK pour JavaScript.

```
/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating set operations in DynamoDB");

  try {
    // Example 1: Create a string set
    console.log("\nExample 1: Creating a string set");
    const response1 = await createSet(
      config,
      tableName,
      key,
      "Interests",
      ["Reading", "Hiking", "Cooking"],
      "string"
    );

    console.log("Created set:", response1.Attributes);

    // Example 2: Add elements to a set
    console.log("\nExample 2: Adding elements to a set");
    const response2 = await addToSet(
      config,
      tableName,
      key,
      "Interests",
      ["Photography", "Travel"],
      "string"
    );

    console.log("Updated set after adding elements:", response2.Attributes);

    // Example 3: Remove elements from a set
    console.log("\nExample 3: Removing elements from a set");
    const response3 = await removeFromSet(
      config,
      tableName,
```

```
    key,
    "Interests",
    ["Cooking"],
    "string"
  );

  console.log("Updated set after removing elements:", response3.Attributes);

  // Example 4: Create a number set
  console.log("\nExample 4: Creating a number set");
  const response4 = await createSet(
    config,
    tableName,
    key,
    "FavoriteNumbers",
    [7, 42, 99],
    "number"
  );

  console.log("Created number set:", response4.Attributes);

  // Example 5: Replace an entire set
  console.log("\nExample 5: Replacing an entire set");
  const response5 = await replaceSet(
    config,
    tableName,
    key,
    "Interests",
    ["Gaming", "Movies", "Music"],
    "string"
  );

  console.log("Replaced set:", response5.Attributes);

  // Example 6: Remove the last element from a set
  console.log("\nExample 6: Removing the last element from a set");

  // First, create a set with just one element
  await createSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags",
    ["LastTag"],
```

```
      "string"
    );

    // Then, remove the last element
    const response6 = await removeLastElementFromSet(
      config,
      tableName,
      { UserId: "U67890" },
      "Tags"
    );

    console.log(response6.message);
    console.log("Removed value:", response6.removedValue);

    // Get the final state of the items
    console.log("\nFinal state of the items:");
    const item1 = await getItem(config, tableName, key);
    console.log("User U12345:", JSON.stringify(item1, null, 2));

    const item2 = await getItem(config, tableName, { UserId: "U67890" });
    console.log("User U67890:", JSON.stringify(item2, null, 2));

    // Explain set operations
    console.log("\nKey points about set operations in DynamoDB:");
    console.log("1. Use ADD to add elements to a set (duplicates are automatically
 removed)");
    console.log("2. Use DELETE to remove elements from a set");
    console.log("3. Use SET to create a new set or replace an existing one");
    console.log("4. DynamoDB supports three types of sets: string sets, number sets,
 and binary sets");
    console.log("5. When you delete the last element from a set, the attribute
 remains as an empty set");
    console.log("6. To remove an empty set, use the REMOVE operation");
    console.log("7. Sets automatically maintain unique values (no duplicates)");
    console.log("8. You cannot mix data types within a set");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Interrogation d'une table à l'aide de lots d'instructions PartiQL

L'exemple de code suivant illustre comment :

- Obtenez un lot d'éléments en exécutant plusieurs instructions SELECT.
- Ajoutez un lot d'éléments en exécutant plusieurs instructions INSERT.
- Mettez à jour un lot d'éléments en exécutant plusieurs instructions UPDATE.
- Supprimez un lot d'éléments en exécutant plusieurs instructions DELETE.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez des instructions PartiQL par lots.

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

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

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

```
const tableName = "Cities";

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

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
```

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

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

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

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
```

```
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log("Cities inserted.");

  /**
   * Select items.
   */

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

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log("Updated cities.");

  /**
   * Delete the items.
   */

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

  /**
   * Delete the table.
   */

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

```
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Pour plus de détails sur l'API, reportez-vous BatchExecuteStatementà la section Référence des AWS SDK pour JavaScript API.

Interrogation d'une table à l'aide de PartiQL

L'exemple de code suivant illustre comment :

- Obtenez un élément en exécutant une instruction SELECT.
- Ajoutez un élément en exécutant une instruction INSERT.
- Mettez à jour un élément en exécutant une instruction UPDATE.
- Supprimez un élément en exécutant une instruction DELETE.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez des instructions PartiQL uniques.

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

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

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

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

  /**
   * Create a table.
   */
```

```
  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

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

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

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
```

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

  /**
   * Select an item.
   */

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

  /**
   * Update the item.
   */

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

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
```

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

  /**
   * Delete the table.
   */

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

- Pour plus de détails sur l'API, reportez-vous ExecuteStatementà la section Référence des AWS SDK pour JavaScript API.

Interrogation d’une table à l’aide d’un index secondaire global

L’exemple de code suivant montre comment interroger une table à l’aide d’un index secondaire global.

- Interrogez une table DynamoDB à l’aide de sa clé primaire.
- Interrogez un index secondaires globaux (GSI) pour les autres modèles d’accès.
- Comparez les requêtes de table et les requêtes GSI.

SDK pour JavaScript (v3)

Interrogez une table DynamoDB à l'aide de la clé primaire avec. AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
```

```
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}
```

Interrogez un index secondaire global (GSI) DynamoDB avec. AWS SDK pour JavaScript

```
/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
```

```
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation d'une table à l'aide d'une condition begins_with

L'exemple de code suivant montre comment interroger une table à l'aide d'une condition begins_with.

- Utilisez la fonction begins_with dans une expression de condition de clé.
- Filtrez des éléments en fonction d'un modèle de préfixe dans la clé de tri.

SDK pour JavaScript (v3)

Interrogation d'une table DynamoDB à l'aide d'une condition begins_with sur une clé de tri avec le kit AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 prefix
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */
async function queryWithBeginsWith(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
      }
    };
```

```
    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with begins_with: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation d'une table à l'aide d'une plage de dates

L'exemple de code suivant montre comment interroger une table à l'aide d'une plage de dates dans la clé de tri.

- Interrogation d'éléments dans une plage de dates spécifique.
- Utilisez des opérateurs de comparaison sur les clés de tri formatées par date.

SDK pour JavaScript (v3)

Interrogez une table DynamoDB pour les éléments compris dans une plage de dates avec. AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
```

```
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation d'une table avec une expression de filtre complexe

L'exemple de code suivant montre comment interroger une table à l'aide d'une expression de filtre complexe.

- Appliquez des expressions de filtre complexes aux résultats des requêtes.
- Combinez plusieurs conditions à l'aide d'opérateurs logiques.
- Filtrez des éléments en fonction d'attributs non essentiels.

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec une expression de filtre complexe à l'aide de. AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
```

```
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation d'une table avec une expression de filtre dynamique

L'exemple de code suivant montre comment interroger une table à l'aide d'une expression de filtre dynamique.

- Créez des expressions de filtre de manière dynamique lors de l'exécution.
- Construction de conditions de filtre en fonction des données saisies par l'utilisateur ou de l'état de l'application.

- Ajoutez ou supprimez des critères de filtre sous certaines conditions.

SDK pour JavaScript (v3)

Interrogez une table DynamoDB à l'aide d'une expression de filtre construite dynamiquement à l'aide de. AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
```

```
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation d'une table avec des attributs imbriqués

L'exemple de code suivant montre comment interroger une table avec des attributs imbriqués.

- Accédez aux éléments DynamoDB et filtrez en fonction des attributs imbriqués.

- Utilisez des expressions de chemin de document pour référencer des éléments imbriqués.

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec des attributs imbriqués à l'aide de. AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
async function queryWithNestedAttribute(
  config,
  tableName,
  productId,
  category
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "product_id = :productId",
      FilterExpression: "details.category = :category",
      ExpressionAttributeValues: {
        ":productId": { S: productId },
        ":category": { S: category }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
```

```
    console.error(`Error querying with nested attribute: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation d'une table avec pagination

L'exemple de code suivant montre comment interroger une table avec pagination.

- Mise en œuvre de la pagination pour les résultats des requêtes DynamoDB.
- Utilisez le LastEvaluatedKey pour récupérer les pages suivantes.
- Contrôlez le nombre d'éléments par page à l'aide du paramètre Limit.

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec la pagination à l'aide de. AWS SDK pour JavaScript

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
```

```
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
```

```
      console.log(`Processing page ${pageCount} with ${response.Items.length}
 items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in ${pageCount}
 pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */
```

```
module.exports = { queryWithPagination };
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation d'une table avec des lectures fortement cohérentes

L'exemple de code suivant montre comment interroger une table avec des lectures fortement cohérentes.

- Configuration du niveau de cohérence pour les requêtes DynamoDB.
- Utilisez des lectures très cohérentes pour obtenir le maximum de up-to-date données.
- Compréhension des compromis entre une cohérence à terme et une forte cohérence.

SDK pour JavaScript (v3)

Interrogez une table DynamoDB avec une cohérence de lecture configurable à l'aide de. AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
```

```
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation des données à l'aide de PartiQL SELECT

L'exemple de code suivant montre comment interroger des données à l'aide des instructions PartiQL SELECT.

SDK pour JavaScript (v3)

Interrogez des éléments d'une table DynamoDB à l'aide des instructions PartiQL SELECT avec. AWS SDK pour JavaScript

```
/**
 * This example demonstrates how to query items from a DynamoDB table using PartiQL.
 * It shows different ways to select data with various index types.
```

```
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select an item by its primary key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
```

```
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
```

```
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select items using a filter condition with PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
```

```
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix
 * @param prefix - The prefix to match
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByPrefix = async (
  tableName: string,
  attributeName: string,
  prefix: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE
 begins_with(${attributeName}, ?)`,
    Parameters: [prefix],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a between condition for range queries.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for range
 * @param startValue - The start value of the range
 * @param endValue - The end value of the range
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
```

```
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ? AND ?
`,
    Parameters: [startValue, endValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");

  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456", "productId",
 "prod789");

  // Select with a filter condition (can use any attribute)
  await selectItemsWithFilter("UsersTable", "userType", "premium");

  // Select items with a prefix (useful for hierarchical data)
  await selectItemsByPrefix("ProductsTable", "category", "electronics");

  // Select items within a range (useful for numeric or date ranges)
  await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01", "2023-12-31");
```

```
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Interrogation des éléments TTL

L'exemple de code suivant montre comment demander des éléments TTL.

SDK pour JavaScript (v3)

Expression filtrée par requête pour rassembler des éléments TTL dans une table DynamoDB à l'aide de. AWS SDK pour JavaScript

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

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1') =>
 {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

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

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

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

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Interrogation de tables à l'aide de modèles de date et d'heure

L'exemple de code suivant montre comment interroger des tables à l'aide de modèles de date et d'heure.

- Stockez et interrogez date/time des valeurs dans DynamoDB.
- Mettez en œuvre des requêtes par plage de dates à l'aide de clés de tri.
- Formattez des chaînes de date pour une interrogation efficace.

SDK pour JavaScript (v3)

Requête utilisant des plages de dates dans les clés de tri avec AWS SDK pour JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
```

```
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
```

```
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

Requête utilisant des variables date-heure avec. AWS SDK pour JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  dateKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
```

```
      KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
 AND :endDate`,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#dateAttr": dateKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range: ${error}`);
    throw error;
  }
}
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour JavaScript .

Compréhension de l'ordre des expressions de mise à jour

L'exemple de code suivant montre comprendre l'ordre des expressions de mise à jour.

- Découverte de la façon dont DynamoDB traite les expressions de mise à jour.
- Comprenez l'ordre des opérations dans les expressions de mise à jour.
- Évitez les résultats inattendus en comprenant l'évaluation des expressions.

SDK pour JavaScript (v3)

Démontrez l'ordre des expressions de mise à jour en utilisant AWS SDK pour JavaScript

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
```

```
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 placeholders
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
```

```
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}

/**
 * Demonstrate that variables hold copies of existing values before modifications.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that uses the values of attributes before they are modified in the same
 expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateValueCopying(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = { ...key, a: 1, b: 2, c: 3 };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));
```

```
  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with an expression that uses values before they are
 modified
  // This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to the
 value of 'b'
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE a SET b = a, c = b",
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Demonstrate the order in which different action types are processed.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the order
 * in which they are processed.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
 counter :increment DELETE set_attr :elements",
```

```
    ExpressionAttributeValues: {
      ":new_value": "Updated value",
      ":increment": 5,
      ":elements": new Set(["B"])
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Update multiple attributes with a single SET action.
 *
 * This function demonstrates how to update multiple attributes in a single SET
 action,
 * which is more efficient than using multiple separate update operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {Object} attributes - The attributes to update and their new values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleSetActions(
  config,
  tableName,
  key,
  attributes
) {
  // Initialize the DynamoDB client
```

```
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET ";
  const expressionAttributeValues = {};

  // Add each attribute to the update expression
  Object.entries(attributes).forEach(([attrName, attrValue], index) => {
    const valuePlaceholder = `:val${index}`;

    if (index > 0) {
      updateExpression += ", ";
    }
    updateExpression += `${attrName} = ${valuePlaceholder}`;

    expressionAttributeValues[valuePlaceholder] = attrValue;
  });

  // Execute the update
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Update an attribute with a value from another attribute or a default value.
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a value
 * from one attribute to another, or use a default value if the source doesn't
 exist.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} sourceAttribute - The attribute to copy the value from
 * @param {string} targetAttribute - The attribute to update
```

```
 * @param {any} defaultValue - The default value to use if the source attribute
 doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
 if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Demonstrate complex update expressions with multiple operations on the same
 attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
 attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
```

```
 */
async function demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    list_attr: [1, 2, 3],
    map_attr: {
      nested1: "value1",
      nested2: "value2"
    }
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple operations on the same attributes
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `
      SET counter = counter + :inc1,
          counter = counter + :inc2,
          map_attr.nested1 = :new_val1,
          map_attr.nested3 = :new_val3,
          list_attr[0] = list_attr[1],
```

```
          list_attr[1] = list_attr[2]
    `,
    ExpressionAttributeValues: {
      ":inc1": 5,
      ":inc2": 3,
      ":new_val1": "updated_value1",
      ":new_val3": "new_value3"
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}
```

Exemple d'utilisation de l'ordre des expressions de mise à jour avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "OrderProcessing";

  console.log("Demonstrating update expression order of operations in DynamoDB");

  try {
    // Example 1: Demonstrating value copying in update expressions
    console.log("\nExample 1: Demonstrating value copying in update expressions");
```

```
  const results1 = await demonstrateValueCopying(
    config,
    tableName,
    { OrderId: "order123" }
  );

  console.log("Initial state:", JSON.stringify(results1.initialState, null, 2));
  console.log("Update response:", JSON.stringify(results1.updateResponse, null,
2));
  console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

  console.log("\nExplanation:");
  console.log("1. The initial state had a=1, b=2, c=3");
  console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:");
  console.log("   - Copied the value of 'a' (which was 1) to be used for 'b'");
  console.log("   - Copied the value of 'b' (which was 2) to be used for 'c'");
  console.log("   - Removed the attribute 'a'");
  console.log("3. The final state has b=1, c=2, and 'a' is removed");
  console.log("4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications");

  // Example 2: Demonstrating the order of different action types
  console.log("\nExample 2: Demonstrating the order of different action types");
  const results2 = await demonstrateActionOrder(
    config,
    tableName,
    { OrderId: "order456" }
  );

  console.log("Initial state:", JSON.stringify(results2.initialState, null, 2));
  console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
  console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

  console.log("\nExplanation:");
  console.log("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE");
  console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
  console.log("3. First, 'to_remove' was removed");
  console.log("4. Then, 'to_modify' was set to a new value");
  console.log("5. Next, 'counter' was incremented by 5");
  console.log("6. Finally, 'B' was removed from the set attribute");
```

```
  // Example 3: Updating multiple attributes in a single SET action
  console.log("\nExample 3: Updating multiple attributes in a single SET action");
  const response3 = await updateWithMultipleSetActions(
    config,
    tableName,
    { OrderId: "order789" },
    {
      Status: "Shipped",
      ShippingDate: "2025-05-28",
      TrackingNumber: "1Z999AA10123456784"
    }
  );

  console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

  // Example 4: Conditional value copying with if_not_exists
  console.log("\nExample 4: Conditional value copying with if_not_exists");
  const response4 = await updateWithConditionalValueCopying(
    config,
    tableName,
    { OrderId: "order101" },
    "PreferredShippingMethod",
    "ShippingMethod",
    "Standard"
  );

  console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

  // Example 5: Multiple operations on the same attribute
  console.log("\nExample 5: Multiple operations on the same attribute");
  const results5 = await demonstrateMultipleOperationsOnSameAttribute(
    config,
    tableName,
    { OrderId: "order202" }
  );

  console.log("Initial state:", JSON.stringify(results5.initialState, null, 2));
  console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
  console.log("Final state:", JSON.stringify(results5.finalState, null, 2));
```

```
    console.log("\nExplanation:");
    console.log("1. The counter was incremented twice (first by 5, then by 3) for a
 total of +8");
    console.log("2. The map attribute had one value updated and a new nested
 attribute added");
    console.log("3. The list attribute had values shifted (value at index 1 moved to
 index 0, value at index 2 moved to index 1)");
    console.log("4. All operations within the SET action are processed from left to
 right");

    // Key points about update expression order of operations
    console.log("\nKey Points About Update Expression Order of Operations:");
    console.log("1. Variables in expressions hold copies of attribute values as they
 existed BEFORE any modifications");
    console.log("2. Multiple actions in an update expression are processed in this
 order: REMOVE, SET, ADD, DELETE");
    console.log("3. Within each action type, operations are processed from left to
 right");
    console.log("4. You can reference the same attribute multiple times in an
 expression");
    console.log("5. You can use if_not_exists() to conditionally set values based on
 attribute existence");
    console.log("6. Using a single update expression with multiple actions is more
 efficient than multiple separate updates");
    console.log("7. The update expression is atomic - either all actions succeed or
 none do");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Mise à jour du paramètre de débit chaud d’une table

L’exemple de code suivant montre comment mettre à jour le paramètre de débit chaud d’une table.

SDK pour JavaScript (v3)

Mettez à jour le paramètre de débit chaud sur une table DynamoDB existante à l'aide du kit AWS SDK pour JavaScript.

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

export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response:
 ${JSON.stringify(response)}`);
    return response;
```

```
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
updateDynamoDBTableWarmThroughput(
  'example-table',
  5, 5,
  'example-index',
  2, 2
);
*/
```

- Pour plus de détails sur l'API, reportez-vous UpdateTableà la section Référence des AWS SDK pour JavaScript API.

Mise à jour du TTL d’un élément

L’exemple de code suivant montre comment mettre à jour la TTL d’un élément.

SDK pour JavaScript (v3)

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

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

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

    const params = {
        TableName: tableName,
        Key: marshall({
```

```
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

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

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-value');
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Mise à jour des données à l’aide de PartiQL UPDATE

L’exemple de code suivant montre comment mettre à jour des données à l’aide des instructions PartiQL UPDATE.

SDK pour JavaScript (v3)

Mettez à jour les éléments d'une table DynamoDB à l'aide des instructions PARTIQL UPDATE avec. AWS SDK pour JavaScript

```
/**
 * This example demonstrates how to update items in a DynamoDB table using PartiQL.
```

```
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateSingleAttribute = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};
```

```
/**
 * Update multiple attributes of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeUpdates - Object containing attribute names and their new values
 * @returns The response from the ExecuteStatementCommand
 */
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");

  // Create parameters array with attribute values followed by the partition key
 value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName} = ?
`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};
```

```
/**
 * Update an item identified by a composite key (partition key + sort key) using
 PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item with a condition to ensure the update only happens if a condition
 is met.
```

```
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
 * Batch update multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param updates - Array of objects containing key and update information
 * @returns The response from the BatchExecuteStatementCommand
```

```
 */
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each update
  const statements = updates.map((update) => {
    return {
      Statement: `UPDATE "${tableName}" SET ${update.attributeName} = ? WHERE
 ${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
  await updateSingleAttribute("UsersTable", "userId", "user123", "email",
 "newemail@example.com");
```

```
// Update multiple attributes at once
await updateMultipleAttributes("UsersTable", "userId", "user123", {
  email: "newemail@example.com",
  name: "John Smith",
  lastLogin: new Date().toISOString(),
});

// Update an item with a composite key (partition key + sort key)
await updateItemWithCompositeKey(
  "OrdersTable",
  "orderId",
  "order456",
  "productId",
  "prod789",
  "quantity",
  5
);

// Update with a condition
await updateItemWithCondition(
  "UsersTable",
  "userId",
  "user123",
  "userStatus",
  "active",
  "userType",
  "premium"
);

// Batch update multiple items
await batchUpdateItems("UsersTable", [
  {
    partitionKeyName: "userId",
    partitionKeyValue: "user123",
    attributeName: "lastLogin",
    attributeValue: new Date().toISOString(),
  },
  {
    partitionKeyName: "userId",
    partitionKeyValue: "user456",
    attributeName: "lastLogin",
    attributeValue: new Date().toISOString(),
  },
]);
```

```
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Utiliser API Gateway pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par Amazon API Gateway.

SDK pour JavaScript (v3)

Montre comment créer une AWS Lambda fonction à l'aide de l'API JavaScript d'exécution Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une fonction Lambda invoquée par Amazon API Gateway qui analyse une table Amazon DynamoDB à la recherche d'anniversaires professionnels et utilise Amazon Simple Notification Service (Amazon SNS) pour envoyer un message texte à vos employés qui les félicitent à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Utilisation d'opérations de compteurs atomiques

L'exemple de code suivant montre comment utiliser les opérations de compteurs atomiques dans DynamoDB.

- Incrémentez les compteurs de manière atomique à l'aide des opérations ADD et SET.
- Incrémentez en toute sécurité les compteurs qui n'existent peut-être pas.
- Mettez en œuvre un verrouillage optimiste pour les opérations de compteur.

SDK pour JavaScript (v3)

Démontrez les opérations de contre-attaque atomique en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 increments.
 * While this approach works, it's less idiomatic for simple increments than using
 ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with an expression
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  defaultValue = 0
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${counterName} = if_not_exists(${counterName}, :default)
 + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement optimistic
 * locking, which prevents race conditions when multiple processes try to update
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithLocking(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  expectedValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
```

```
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ConditionExpression: `${counterName} = :expected`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":expected": expectedValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      data: response
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        error: "Optimistic locking failed: the counter value has changed"
      };
    }
    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not found
 */
async function getCounterValue(
  config,
```

```
  tableName,
  key,
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
    : null;
}
```

Exemple d'utilisation d'opérations de compteur atomique avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use the atomic counter operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;

  console.log("Demonstrating different approaches to increment counters in
 DynamoDB");

  try {
    // Example 1: Using ADD operation (recommended for simple increments)
    console.log("\nExample 1: Incrementing counter with ADD operation");
```

```
    const response1 = await incrementCounterWithAdd(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);

    // Example 2: Using SET operation with an expression
    console.log("\nExample 2: Incrementing counter with SET operation");
    const response2 = await incrementCounterWithSet(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

    // Example 3: Safely incrementing a counter that might not exist
    console.log("\nExample 3: Safely incrementing counter that might not exist");
    const newKey = { ProductId: "P67890" };
    const response3 = await incrementCounterSafely(
      config,
      tableName,
      newKey,
      counterName,
      incrementValue,
      0
    );

    console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

    // Example 4: Incrementing with optimistic locking
    console.log("\nExample 4: Incrementing with optimistic locking");

    // First, get the current counter value
    const currentValue = await getCounterValue(config, tableName, key, counterName);
    console.log(`Current counter value: ${currentValue}`);
```

```
    // Then, try to increment with optimistic locking
    const response4 = await incrementCounterWithLocking(
      config,
      tableName,
      key,
      counterName,
      incrementValue,
      currentValue
    );

    if (response4.success) {
      console.log(`Counter successfully incremented to:
 ${response4.data.Attributes[counterName]}`);
    } else {
      console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
    console.log("2. SET with expressions is more flexible for complex operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might not
 exist");
    console.log("5. Optimistic locking can be added to either approach for
 additional safety");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour JavaScript API.

Utilisation d’opérations conditionnelles

L’exemple de code suivant montre comment utiliser des opérations conditionnelles dans DynamoDB.

- Implémentation d’écritures conditionnelles pour éviter le remplacement de données.
- Utilisez des expressions conditionnelles pour appliquer des règles de gestion.

- Gérez élégamment les échecs des vérifications conditionnelles.

SDK pour JavaScript (v3)

Démontrez les opérations conditionnelles en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  DeleteCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue,
  updateAttribute,
  updateValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
```

```
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateAttribute} = :value`,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":value": updateValue,
      ":condition": conditionValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Condition was met and update was performed",
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, update was not performed",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Perform a conditional delete operation.
 *
 * This function demonstrates how to delete an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to delete
```

```
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalDelete(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the delete parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":condition": conditionValue
    },
    ReturnValues: "ALL_OLD"
  };

  try {
    // Perform the delete operation
    const response = await docClient.send(new DeleteCommand(params));

    return {
      success: true,
      message: "Condition was met and item was deleted",
      deletedItem: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, item was not deleted",
        error: "ConditionalCheckFailedException"
      };
    }
```

```
    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
 * to prevent race conditions when multiple processes update the same item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {Object} updates - The attributes to update
 * @param {number} expectedVersion - The expected current version number
 * @returns {Promise<Object>} - Result of the operation
 */
async function updateWithOptimisticLocking(
  config,
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Add the version update
```

```
  updateExpressions.push("version = :newVersion");

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Update succeeded with optimistic locking",
      newVersion: expectedVersion + 1,
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Optimistic locking failed: the item was modified by another
 process",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 operation).
```

```
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
async function createIfNotExists(
  config,
  tableName,
  item
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Extract the primary key attributes
  const keyAttributes = Object.keys(item).filter(attr =>
    attr === "id" || attr === "ID" || attr === "Id" ||
    attr.endsWith("Id") || attr.endsWith("ID") ||
    attr.endsWith("Key")
  );

  if (keyAttributes.length === 0) {
    throw new Error("Could not determine primary key attributes");
  }

  // Create a condition expression that checks if the item doesn't exist
  const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

  // Define the put parameters with a condition expression
  const params = {
    TableName: tableName,
    Item: item,
    ConditionExpression: conditionExpression
  };

  try {
    // Perform the put operation
    await docClient.send(new PutCommand(params));

    return {
      success: true,
      message: "Item was created because it didn't exist",
      item
```

```
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Item already exists, creation was skipped",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));
```

```
  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemple d'utilisation d'opérations conditionnelles avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use conditional operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating conditional operations in DynamoDB");

  try {
    // Example 1: Conditional update based on attribute value
    console.log("\nExample 1: Conditional update based on attribute value");
    const updateResult = await conditionalUpdate(
      config,
      tableName,
      key,
      "Category",
      "Electronics",
      "Price",
      299.99
    );

    console.log(`Result: ${updateResult.message}`);
    if (updateResult.success) {
      console.log("Updated attributes:", updateResult.updatedAttributes);
    }

    // Example 2: Conditional delete based on attribute value
    console.log("\nExample 2: Conditional delete based on attribute value");
    const deleteResult = await conditionalDelete(
      config,
      tableName,
      key,
      "InStock",
```

```
    false
  );

  console.log(`Result: ${deleteResult.message}`);
  if (deleteResult.success) {
    console.log("Deleted item:", deleteResult.deletedItem);
  }

  // Example 3: Optimistic locking with version number
  console.log("\nExample 3: Optimistic locking with version number");

  // First, get the current item to check its version
  const currentItem = await getItem(config, tableName, { ProductId: "P67890" });
  const currentVersion = currentItem ? (currentItem.version || 0) : 0;

  console.log(`Current version: ${currentVersion}`);

  // Then, update with optimistic locking
  const lockingResult = await updateWithOptimisticLocking(
    config,
    tableName,
    { ProductId: "P67890" },
    {
      Name: "Updated Product Name",
      Description: "This is an updated description"
    },
    currentVersion
  );

  console.log(`Result: ${lockingResult.message}`);
  if (lockingResult.success) {
    console.log(`New version: ${lockingResult.newVersion}`);
    console.log("Updated attributes:", lockingResult.updatedAttributes);
  }

  // Example 4: Create item only if it doesn't exist
  console.log("\nExample 4: Create item only if it doesn't exist");
  const createResult = await createIfNotExists(
    config,
    tableName,
    {
      ProductId: "P99999",
      Name: "New Product",
      Category: "Accessories",
```

```
        Price: 19.99,
        InStock: true
      }
    );

    console.log(`Result: ${createResult.message}`);
    if (createResult.success) {
      console.log("Created item:", createResult.item);
    }

    // Explain conditional operations
    console.log("\nKey points about conditional operations:");
    console.log("1. Conditional operations only succeed if the condition is met");
    console.log("2. ConditionalCheckFailedException indicates the condition wasn't
 met");
    console.log("3. Optimistic locking prevents race conditions in concurrent
 updates");
    console.log("4. attribute_exists and attribute_not_exists are useful for
 checking if attributes are present");
    console.log("5. Conditional operations are atomic - they either succeed
 completely or fail completely");
    console.log("6. You can use any valid comparison operators and functions in
 condition expressions");
    console.log("7. Conditional operations don't consume write capacity if the
 condition fails");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - DeleteItem
  - PutItem
  - UpdateItem

Utilisation des noms d’attributs d’expression

L’exemple de code suivant montre comment utiliser les noms d’attributs d’expression dans DynamoDB.

- Utilisez des mots réservés dans les expressions DynamoDB.
- Utilisez des espaces réservés pour les noms d’attributs d’expression.
- Gérez des caractères spéciaux dans les noms d’attributs.

SDK pour JavaScript (v3)

Démontrez les noms d'attributs d'expression en utilisant AWS SDK pour JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  QueryCommand,
  ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateReservedWordAttribute(
  config,
  tableName,
  key,
  reservedWordAttribute,
  value
) {
```

```
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters to
 update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
  specialCharAttribute,
  value
```

```
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": specialCharAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
 * @param {string} reservedWordAttribute - The reserved word attribute to filter on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
```

```
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the query parameters using expression attribute names
  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pkName = :pkValue",
    FilterExpression: "#attr = :value",
    ExpressionAttributeNames: {
      "#pkName": partitionKeyName,
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":pkValue": partitionKeyValue,
      ":value": value
    }
  };

  // Perform the query operation
  const response = await docClient.send(new QueryCommand(params));

  return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedReservedWordAttribute(
  config,
  tableName,
```

```
  key,
  attributePath,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names for each part of the path
  const expressionAttributeNames = {};
  for (let i = 0; i < attributePath.length; i++) {
    expressionAttributeNames[`#attr${i}`] = attributePath[i];
  }

  // Build the attribute path using the expression attribute names
  const attributePathExpression = attributePath
    .map((_, i) => `#attr${i}`)
    .join(".");

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
 * in a complex filter expression.
 *
 * @param {Object} config - AWS configuration object
```

```
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function scanWithMultipleAttributeNames(
  config,
  tableName,
  filters
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names and values
  const expressionAttributeNames = {};
  const expressionAttributeValues = {};
  const filterConditions = [];

  // Build the filter expression
  Object.entries(filters).forEach(([attrName, value], index) => {
    const nameKey = `#attr${index}`;
    const valueKey = `:val${index}`;

    expressionAttributeNames[nameKey] = attrName;
    expressionAttributeValues[valueKey] = value;
    filterConditions.push(`${nameKey} = ${valueKey}`);
  });

  // Join the filter conditions with AND
  const filterExpression = filterConditions.join(" AND ");

  // Define the scan parameters
  const params = {
    TableName: tableName,
    FilterExpression: filterExpression,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Perform the scan operation
  const response = await docClient.send(new ScanCommand(params));

  return response;
}
```

```
/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Exemple d'utilisation de noms d'attributs d'expression avec AWS SDK pour JavaScript.

```
/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
```

```
const key = { ProductId: "P12345" };

console.log("Demonstrating expression attribute names in DynamoDB");

try {
  // Example 1: Update an attribute that is a reserved word
  console.log("\nExample 1: Updating an attribute that is a reserved word");
  const response1 = await updateReservedWordAttribute(
    config,
    tableName,
    key,
    "Size", // "SIZE" is a reserved word in DynamoDB
    "Large"
  );

  console.log("Updated attribute:", response1.Attributes);

  // Example 2: Update an attribute with special characters
  console.log("\nExample 2: Updating an attribute with special characters");
  const response2 = await updateSpecialCharacterAttribute(
    config,
    tableName,
    key,
    "Product-Type", // Contains a hyphen, which is a special character
    "Electronics"
  );

  console.log("Updated attribute:", response2.Attributes);

  // Example 3: Query with a reserved word attribute
  console.log("\nExample 3: Querying with a reserved word attribute");
  const response3 = await queryWithReservedWordAttribute(
    config,
    tableName,
    "Category",
    "Electronics",
    "Count", // "COUNT" is a reserved word in DynamoDB
    10
  );

  console.log(`Found ${response3.Items.length} items`);

  // Example 4: Update a nested attribute with reserved words in the path
```

```
    console.log("\nExample 4: Updating a nested attribute with reserved words in the
path");
    const response4 = await updateNestedReservedWordAttribute(
      config,
      tableName,
      key,
      ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
      30
    );

    console.log("Updated nested attribute:", response4.Attributes);

    // Example 5: Scan with multiple attribute name placeholders
    console.log("\nExample 5: Scanning with multiple attribute name placeholders");
    const response5 = await scanWithMultipleAttributeNames(
      config,
      tableName,
      {
        "Size": "Large",
        "Count": 10,
        "Product-Type": "Electronics"
      }
    );

    console.log(`Found ${response5.Items.length} items`);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Show some common reserved words
    console.log("\nSome common DynamoDB reserved words:");
    const commonReservedWords = [
      "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
      "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
      "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
      "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
      "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN", "NAME",
      "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
      "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
      "VIEW", "WHERE"
    ];
    console.log(commonReservedWords.join(", "));
```

```
    // Explain expression attribute names
    console.log("\nKey points about expression attribute names:");
    console.log("1. Use expression attribute names (#name) for reserved words");
    console.log("2. Use expression attribute names for attributes with special
 characters");
    console.log("3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
    console.log("4. Expression attribute names are required for nested attributes
 with reserved words");
    console.log("5. You can use multiple expression attribute names in a single
 expression");
    console.log("6. Expression attribute names are case-sensitive");
    console.log("7. Expression attribute names are only used in expressions, not in
 the actual data");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript .
  - Interrogation
  - UpdateItem

Utilisent des événements planifiés pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par un événement EventBridge planifié par Amazon.

SDK pour JavaScript (v3)

> Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Exemples sans serveur

Invocation d'une fonction Lambda à partir d'un déclencheur DynamoDB

L'exemple de code suivant montre comment mettre en œuvre une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux DynamoDB. La fonction récupère les données utiles DynamoDB et journalise le contenu de l'enregistrement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. 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)}`);
};
```

Consommation d'un événement DynamoDB avec Lambda en utilisant. 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)}`);
};
```

Signalement des échecs d’articles par lots pour les fonctions Lambda à l’aide d’un déclencheur DynamoDB

L’exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d’un flux DynamoDB. La fonction signale les défaillances échecs d’articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. 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: [] };
};
```

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. TypeScript

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

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

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

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
```

```
};
```

EC2 Exemples Amazon utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la AWS SDK pour JavaScript version (v3) avec Amazon EC2.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour Amazon EC2

L'exemple de code suivant montre comment commencer à utiliser Amazon EC2.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  const client = new EC2Client();
  try {
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({}),
    );

    const securityGroupList = SecurityGroups.slice(0, 9)
      .map((sg) => ` • ${sg.GroupId}: ${sg.GroupName}`)
      .join("\n");

    console.log(
      "Hello, Amazon EC2! Let's list up to 10 of your security groups:",
    );
    console.log(securityGroupList);
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeSecurityGroupsà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez une paire de clés et un groupe de sécurité.
- Sélectionnez une Amazon Machine Image (AMI) et un type d'instance compatible, puis créez une instance.
- Arrêtez l'instance, puis redémarrez-la.
- Associez une adresse IP Elastic à votre instance
- Connectez-vous à votre instance avec SSH, puis nettoyez les ressources.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Ce fichier contient une liste des actions courantes utilisées avec EC2. Les étapes sont construites à l'aide d'un framework de scénarios qui simplifie l'exécution d'un exemple interactif. Pour le contexte complet, consultez le GitHub référentiel.

```
import { tmpdir } from "node:os";
import { writeFile, mkdtemp, rm } from "node:fs/promises";
import { join } from "node:path";
import { get } from "node:http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
  DeleteSecurityGroupCommand,
  DisassociateAddressCommand,
```

```
  paginateDescribeImages,
  paginateDescribeInstances,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

/**
 * @typedef {{
 *   ec2Client: import('@aws-sdk/client-ec2').EC2Client,
 *   errors: Error[],
 *   keyPairId?: string,
 *   tmpDirectory?: string,
 *   securityGroupId?: string,
 *   ipAddress?: string,
 *   images?: import('@aws-sdk/client-ec2').Image[],
 *   image?: import('@aws-sdk/client-ec2').Image,
 *   instanceTypes?: import('@aws-sdk/client-ec2').InstanceTypeInfo[],
 *   instanceId?: string,
 *   instanceIpAddress?: string,
 *   allocationId?: string,
 *   allocatedIpAddress?: string,
 *   associationId?: string,
 * }} State
 */

/**
 * A skip function provided to the `skipWhen` of a Step when you want
 * to ignore that step if any errors have occurred.
 * @param {State} state
```

```
 */
const skipWhenErrors = (state) => state.errors.length > 0;

const MAX_WAITER_TIME_IN_SECONDS = 60 * 8;

export const confirm = new ScenarioInput("confirmContinue", "Continue?", {
  type: "confirm",
  skipWhen: skipWhenErrors,
});

export const exitOnNoConfirm = new ScenarioAction(
  "exitOnConfirmContinueFalse",
  (/** @type { { earlyExit: boolean } & Record<string, any>} */ state) => {
    if (!state[confirm.name]) {
      state.earlyExit = true;
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

export const greeting = new ScenarioOutput(
  "greeting",
  `

Welcome to the Amazon EC2 basic usage scenario.

Before you launch an instances, you'll need to provide a few things:
 - A key pair - This is for SSH access to your EC2 instance. You only need to
 provide the name.
 - A security group - This is used for configuring access to your instance. Again,
 only the name is needed.
 - An IP address - Your public IP address will be fetched.
 - An Amazon Machine Image (AMI)
 - A compatible instance type`,
  { header: true, preformatted: true, skipWhen: skipWhenErrors },
);

export const provideKeyPairName = new ScenarioInput(
  "keyPairName",
  "Provide a name for a new key pair.",
  { type: "input", default: "ec2-example-key-pair", skipWhen: skipWhenErrors },
);
```

```
export const createKeyPair = new ScenarioAction(
  "createKeyPair",
  async (/** @type {State} */ state) => {
    try {
      // Create a key pair in Amazon EC2.
      const { KeyMaterial, KeyPairId } = await state.ec2Client.send(
        // A unique name for the key pair. Up to 255 ASCII characters.
        new CreateKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
      );

      state.keyPairId = KeyPairId;

      // Save the private key in a temporary location.
      state.tmpDirectory = await mkdtemp(join(tmpdir(), "ec2-scenario-tmp"));
      await writeFile(
        `${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem`,
        KeyMaterial,
        {
          mode: 0o400,
        },
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidKeyPair.Duplicate"
      ) {
        caught.message = `${caught.message}. Try another key name.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logKeyPair = new ScenarioOutput(
  "logKeyPair",
  (/** @type {State} */ state) =>
    `Created the key pair ${state[provideKeyPairName.name]}.`,
  { skipWhen: skipWhenErrors },
);

export const confirmDeleteKeyPair = new ScenarioInput(
```

```
  "confirmDeleteKeyPair",
  "Do you want to delete the key pair?",
  {
    type: "confirm",
    // Don't do anything when a key pair was never created.
    skipWhen: (/** @type {State} */ state) => !state.keyPairId,
  },
);

export const maybeDeleteKeyPair = new ScenarioAction(
  "deleteKeyPair",
  async (/** @type {State} */ state) => {
    try {
      // Delete a key pair by name from EC2
      await state.ec2Client.send(
        new DeleteKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        // Occurs when a required parameter (e.g. KeyName) is undefined.
        caught.name === "MissingParameter"
      ) {
        caught.message = `${caught.message}. Did you provide the required value?`;
      }
      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no key pair to delete or the user chooses
    // to keep it.
    skipWhen: (/** @type {State} */ state) =>
      !state.keyPairId || !state[confirmDeleteKeyPair.name],
  },
);

export const provideSecurityGroupName = new ScenarioInput(
  "securityGroupName",
  "Provide a name for a new security group.",
  { type: "input", default: "ec2-scenario-sg", skipWhen: skipWhenErrors },
);

export const createSecurityGroup = new ScenarioAction(
  "createSecurityGroup",
```

```
  async (/** @type {State} */ state) => {
    try {
      // Create a new security group that will be used to configure ingress/egress
 for
      // an EC2 instance.
      const { GroupId } = await state.ec2Client.send(
        new CreateSecurityGroupCommand({
          GroupName: state[provideSecurityGroupName.name],
          Description: "A security group for the Amazon EC2 example.",
        }),
      );
      state.securityGroupId = GroupId;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidGroup.Duplicate") {
        caught.message = `${caught.message}. Please provide a different name for
 your security group.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSecurityGroup = new ScenarioOutput(
  "logSecurityGroup",
  (/** @type {State} */ state) =>
    `Created the security group ${state.securityGroupId}.`,
  { skipWhen: skipWhenErrors },
);

export const confirmDeleteSecurityGroup = new ScenarioInput(
  "confirmDeleteSecurityGroup",
  "Do you want to delete the security group?",
  {
    type: "confirm",
    // Don't do anything when a security group was never created.
    skipWhen: (/** @type {State} */ state) => !state.securityGroupId,
  },
);

export const maybeDeleteSecurityGroup = new ScenarioAction(
  "deleteSecurityGroup",
  async (/** @type {State} */ state) => {
```

```
    try {
      // Delete the security group if the 'skipWhen' condition below is not met.
      await state.ec2Client.send(
        new DeleteSecurityGroupCommand({
          GroupId: state.securityGroupId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidGroupId.Malformed"
      ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
      }
      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no security group to delete
    // or the user chooses to keep it.
    skipWhen: (/** @type {State} */ state) =>
      !state.securityGroupId || !state[confirmDeleteSecurityGroup.name],
  },
);

export const authorizeSecurityGroupIngress = new ScenarioAction(
  "authorizeSecurity",
  async (/** @type {State} */ state) => {
    try {
      // Get the public IP address of the machine running this example.
      const ipAddress = await new Promise((res, rej) => {
        get("http://checkip.amazonaws.com", (response) => {
          let data = "";
          response.on("data", (chunk) => {
            data += chunk;
          });
          response.on("end", () => res(data.trim()));
        }).on("error", (err) => {
          rej(err);
        });
      });
      state.ipAddress = ipAddress;
      // Allow ingress from the IP address above to the security group.
      // This will allow you to SSH into the EC2 instance.
```

```
      const command = new AuthorizeSecurityGroupIngressCommand({
        GroupId: state.securityGroupId,
        IpPermissions: [
          {
            IpProtocol: "tcp",
            FromPort: 22,
            ToPort: 22,
            IpRanges: [{ CidrIp: `${ipAddress}/32` }],
          },
        ],
      });

      await state.ec2Client.send(command);
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidGroupId.Malformed"
      ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSecurityGroupIngress = new ScenarioOutput(
  "logSecurityGroupIngress",
  (/** @type {State} */ state) =>
    `Allowed SSH access from your public IP: ${state.ipAddress}.`,
  { skipWhen: skipWhenErrors },
);

export const getImages = new ScenarioAction(
  "images",
  async (/** @type {State} */ state) => {
    const AMIs = [];
    // Some AWS services publish information about common artifacts as AWS Systems
 Manager (SSM)
    // public parameters. For example, the Amazon Elastic Compute Cloud (Amazon EC2)
    // service publishes information about Amazon Machine Images (AMIs) as public
 parameters.
```

```
  // Create the paginator for getting images. Actions that return multiple pages
of
  // results have paginators to simplify those calls.
  const getParametersByPathPaginator = paginateGetParametersByPath(
    {
      // Not storing this client in state since it's only used once.
      client: new SSMClient({}),
    },
    {
      // The path to the public list of the latest amazon-linux instances.
      Path: "/aws/service/ami-amazon-linux-latest",
    },
  );

  try {
    for await (const page of getParametersByPathPaginator) {
      for (const param of page.Parameters) {
        // Filter by Amazon Linux 2
        if (param.Name.includes("amzn2")) {
          AMIs.push(param.Value);
        }
      }
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidFilterValue") {
      caught.message = `${caught.message} Please provide a valid filter value for
paginateGetParametersByPath.`;
    }
    state.errors.push(caught);
    return;
  }

  const imageDetails = [];
  const describeImagesPaginator = paginateDescribeImages(
    { client: state.ec2Client },
    // The images found from the call to SSM.
    { ImageIds: AMIs },
  );

  try {
    // Get more details for the images found above.
    for await (const page of describeImagesPaginator) {
      imageDetails.push(...(page.Images || []));
    }
```

```
      // Store the image details for later use.
      state.images = imageDetails;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidAMIID.NotFound") {
        caught.message = `${caught.message}. Please provide a valid image id.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const provideImage = new ScenarioInput(
  "image",
  "Select one of the following images.",
  {
    type: "select",
    choices: (/** @type { State } */ state) =>
      state.images.map((image) => ({
        name: `${image.Description}`,
        value: image,
      })),
    default: (/** @type { State } */ state) => state.images[0],
    skipWhen: skipWhenErrors,
  },
);

export const getCompatibleInstanceTypes = new ScenarioAction(
  "getCompatibleInstanceTypes",
  async (/** @type {State} */ state) => {
    // Get more details about instance types that match the architecture of
    // the provided image.
    const paginator = paginateDescribeInstanceTypes(
      { client: state.ec2Client, pageSize: 25 },
      {
        Filters: [
          {
            Name: "processor-info.supported-architecture",
            // The value selected from provideImage()
            Values: [state.image.Architecture],
          },
          // Filter for smaller, less expensive, types.
```

```
          { Name: "instance-type", Values: ["*.micro", "*.small"] },
        ],
      },
    );

    const instanceTypes = [];

    try {
      for await (const page of paginator) {
        if (page.InstanceTypes.length) {
          instanceTypes.push(...(page.InstanceTypes || []));
        }
      }

      if (!instanceTypes.length) {
        state.errors.push(
          "No instance types matched the instance type filters.",
        );
      }
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check the provided values and
 try again.`;
      }

      state.errors.push(caught);
    }

    state.instanceTypes = instanceTypes;
  },
  { skipWhen: skipWhenErrors },
);

export const provideInstanceType = new ScenarioInput(
  "instanceType",
  "Select an instance type.",
  {
    choices: (/** @type {State} */ state) =>
      state.instanceTypes.map((instanceType) => ({
        name: `${instanceType.InstanceType} - Memory:
${instanceType.MemoryInfo.SizeInMiB}`,
        value: instanceType.InstanceType,
      })),
    type: "select",
```

```
    default: (/** @type {State} */ state) =>
      state.instanceTypes[0].InstanceType,
    skipWhen: skipWhenErrors,
  },
);

export const runInstance = new ScenarioAction(
  "runInstance",
  async (/** @type { State } */ state) => {
    const { Instances } = await state.ec2Client.send(
      new RunInstancesCommand({
        KeyName: state[provideKeyPairName.name],
        SecurityGroupIds: [state.securityGroupId],
        ImageId: state.image.ImageId,
        InstanceType: state[provideInstanceType.name],
        // Availability Zones have capacity limitations that may impact your ability
 to launch instances.
        // The `RunInstances` operation will only succeed if it can allocate at
 least the `MinCount` of instances.
        // However, EC2 will attempt to launch up to the `MaxCount` of instances,
 even if the full request cannot be satisfied.
        // If you need a specific number of instances, use `MinCount` and `MaxCount`
 set to the same value.
        // If you want to launch up to a certain number of instances, use `MaxCount`
 and let EC2 provision as many as possible.
        // If you require a minimum number of instances, but do not want to exceed a
 maximum, use both `MinCount` and `MaxCount`.
        MinCount: 1,
        MaxCount: 1,
      }),
    );

    state.instanceId = Instances[0].InstanceId;

    try {
      // Poll `DescribeInstanceStatus` until status is "ok".
      await waitUntilInstanceStatusOk(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [Instances[0].InstanceId] },
      );
    } catch (caught) {
```

```
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logRunInstance = new ScenarioOutput(
  "logRunInstance",
  "The next step is to run your EC2 instance for the first time. This can take a few
 minutes.",
  { header: true, skipWhen: skipWhenErrors },
);

export const describeInstance = new ScenarioAction(
  "describeInstance",
  async (/** @type { State } */ state) => {
    /** @type { import("@aws-sdk/client-ec2").Instance[] } */
    const instances = [];

    try {
      const paginator = paginateDescribeInstances(
        {
          client: state.ec2Client,
        },
        {
          // Only get our created instance.
          InstanceIds: [state.instanceId],
        },
      );

      for await (const page of paginator) {
        for (const reservation of page.Reservations) {
          instances.push(...reservation.Instances);
        }
      }
      if (instances.length !== 1) {
        throw new Error(`Instance ${state.instanceId} not found.`);
      }
```

```
      // The only info we need is the IP address for SSH purposes.
      state.instanceIpAddress = instances[0].PublicIpAddress;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check provided values and try
 again.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSSHConnectionInfo = new ScenarioOutput(
  "logSSHConnectionInfo",
  (/** @type { State } */ state) =>
    `You can now SSH into your instance using the following command:
ssh -i ${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem ec2-user@
${state.instanceIpAddress}`,
  { preformatted: true, skipWhen: skipWhenErrors },
);

export const logStopInstance = new ScenarioOutput(
  "logStopInstance",
  "Stopping your EC2 instance.",
  { skipWhen: skipWhenErrors },
);

export const stopInstance = new ScenarioAction(
  "stopInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StopInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStopped(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
```

```
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  // Don't try to stop an instance that doesn't exist.
  { skipWhen: (/** @type { State } */ state) => !state.instanceId },
);

export const logIpAddressBehavior = new ScenarioOutput(
  "logIpAddressBehavior",
  [
    "When you run an instance, by default it's assigned an IP address.",
    "That IP address is not static. It will change every time the instance is
 restarted.",
    "The next step is to stop and restart your instance to demonstrate this
 behavior.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const logStartInstance = new ScenarioOutput(
  "logStartInstance",
  (/** @type { State } */ state) => `Starting instance ${state.instanceId}`,
  { skipWhen: skipWhenErrors },
);

export const startInstance = new ScenarioAction(
  "startInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StartInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStatusOk(
```

```
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logIpAllocation = new ScenarioOutput(
  "logIpAllocation",
  [
    "It is possible to have a static IP address.",
    "To demonstrate this, an IP will be allocated and associated to your EC2
 instance.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const allocateIp = new ScenarioAction(
  "allocateIp",
  async (/** @type { State } */ state) => {
    try {
      // An Elastic IP address is allocated to your AWS account, and is yours until
 you release it.
      const { AllocationId, PublicIp } = await state.ec2Client.send(
        new AllocateAddressCommand({}),
      );
      state.allocationId = AllocationId;
      state.allocatedIpAddress = PublicIp;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        caught.message = `${caught.message}. Did you provide these values?`;
      }
      state.errors.push(caught);
```

```
    }
  },
  { skipWhen: skipWhenErrors },
);

export const associateIp = new ScenarioAction(
  "associateIp",
  async (/** @type { State } */ state) => {
    try {
      // Associate an allocated IP address to an EC2 instance. An IP address can be
 allocated
      // with the AllocateAddress action.
      const { AssociationId } = await state.ec2Client.send(
        new AssociateAddressCommand({
          AllocationId: state.allocationId,
          InstanceId: state.instanceId,
        }),
      );
      state.associationId = AssociationId;
      // Update the IP address that is being tracked to match
      // the one just associated.
      state.instanceIpAddress = state.allocatedIpAddress;
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Did you provide the ID of a valid
 Elastic IP address AllocationId?`;
      }
      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logStaticIpProof = new ScenarioOutput(
  "logStaticIpProof",
  "The IP address should remain the same even after stopping and starting the
 instance.",
  { header: true, skipWhen: skipWhenErrors },
);

export const logCleanUp = new ScenarioOutput(
```

```
  "logCleanUp",
  "That's it! You can choose to clean up the resources now, or clean them up on your
 own later.",
  { header: true, skipWhen: skipWhenErrors },
);

export const confirmDisassociateAddress = new ScenarioInput(
  "confirmDisassociateAddress",
  "Do you want to disassociate and release the static IP address created earlier?",
  {
    type: "confirm",
    skipWhen: (/** @type { State } */ state) => !state.associationId,
  },
);

export const maybeDisassociateAddress = new ScenarioAction(
  "maybeDisassociateAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new DisassociateAddressCommand({
          AssociationId: state.associationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAssociationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid association
 ID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.associationId,
  },
);

export const maybeReleaseAddress = new ScenarioAction(
  "maybeReleaseAddress",
  async (/** @type { State } */ state) => {
```

```
    try {
      await state.ec2Client.send(
        new ReleaseAddressCommand({
          AllocationId: state.allocationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid AllocationID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.allocationId,
  },
);

export const confirmTerminateInstance = new ScenarioInput(
  "confirmTerminateInstance",
  "Do you want to terminate the instance?",
  // Don't do anything when an instance was never run.
  {
    skipWhen: (/** @type { State } */ state) => !state.instanceId,
    type: "confirm",
  },
);

export const maybeTerminateInstance = new ScenarioAction(
  "terminateInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new TerminateInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );
      await waitUntilInstanceTerminated(
        { client: state.ec2Client },
        { InstanceIds: [state.instanceId] },
```

```
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
 waiter.`;
      }

      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no instance to terminate or the
    // use chooses not to terminate.
    skipWhen: (/** @type { State } */ state) =>
      !state.instanceId || !state[confirmTerminateInstance.name],
  },
);

export const deleteTemporaryDirectory = new ScenarioAction(
  "deleteTemporaryDirectory",
  async (/** @type { State } */ state) => {
    try {
      await rm(state.tmpDirectory, { recursive: true });
    } catch (caught) {
      state.errors.push(caught);
    }
  },
);

export const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State}*/ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    preformatted: true,
    header: true,
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - AllocateAddress
  - AssociateAddress
  - AuthorizeSecurityGroupIngress
  - CreateKeyPair
  - CreateSecurityGroup
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DescribeImages
  - DescribeInstanceTypes
  - DescribeInstances
  - DescribeKeyPairs
  - DescribeSecurityGroups
  - DisassociateAddress
  - ReleaseAddress
  - RunInstances
  - StartInstances
  - StopInstances
  - TerminateInstances
  - UnmonitorInstances

Actions

`AllocateAddress`

L'exemple de code suivant montre comment utiliser`AllocateAddress`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { AllocateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Allocates an Elastic IP address to your AWS account.
 */
export const main = async () => {
  const client = new EC2Client({});
  const command = new AllocateAddressCommand({});

  try {
    const { AllocationId, PublicIp } = await client.send(command);
    console.log("A new IP address has been allocated to your account:");
    console.log(`ID: ${AllocationId} Public IP: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2.
 Look under Network & Security > Elastic IPs",
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
import { fileURLToPath } from "node:url";
// Call function if run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Pour plus de détails sur l'API, reportez-vous AllocateAddressà la section Référence des AWS SDK pour JavaScript API.

`AssociateAddress`

L'exemple de code suivant montre comment utiliser`AssociateAddress`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { AssociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Associates an Elastic IP address, or carrier IP address (for instances that are
 in subnets in Wavelength Zones)
 * with an instance or a network interface.
 * @param {{ instanceId: string, allocationId: string }} options
 */
export const main = async ({ instanceId, allocationId }) => {
  const client = new EC2Client({});
  const command = new AssociateAddressCommand({
    // You need to allocate an Elastic IP address before associating it with an
 instance.
    // You can do that with the AllocateAddressCommand.
    AllocationId: allocationId,
    // You need to create an EC2 instance before an IP address can be associated
 with it.
    // You can do that with the RunInstancesCommand.
    InstanceId: instanceId,
  });

  try {
    const { AssociationId } = await client.send(command);
    console.log(
      `Address with allocation ID ${allocationId} is now associated with instance
 ${instanceId}.`,
      `The association ID is ${AssociationId}.`,
```

```
    );
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Did you provide the ID of a valid Elastic IP address
 AllocationId?`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous AssociateAddressà la section Référence des AWS SDK pour JavaScript API.

`AuthorizeSecurityGroupIngress`

L'exemple de code suivant montre comment utiliser`AuthorizeSecurityGroupIngress`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  AuthorizeSecurityGroupIngressCommand,
  EC2Client,
} from "@aws-sdk/client-ec2";

/**
 * Adds the specified inbound (ingress) rules to a security group.
 * @param {{ groupId: string, ipAddress: string }} options
 */
```

```
export const main = async ({ groupId, ipAddress }) => {
  const client = new EC2Client({});
  const command = new AuthorizeSecurityGroupIngressCommand({
    // Use a group ID from the AWS console or
    // the DescribeSecurityGroupsCommand.
    GroupId: groupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        // The IP address to authorize.
        // For more information on this notation, see
        // https://en.wikipedia.org/wiki/Classless_Inter-
Domain_Routing#CIDR_notation
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  try {
    const { SecurityGroupRules } = await client.send(command);
    console.log(JSON.stringify(SecurityGroupRules, null, 2));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous AuthorizeSecurityGroupIngressà la section Référence des AWS SDK pour JavaScript API.

`CreateKeyPair`

L'exemple de code suivant montre comment utiliser`CreateKeyPair`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates an ED25519 or 2048-bit RSA key pair with the specified name and in the
 specified PEM or PPK format.
 * Amazon EC2 stores the public key and displays the private key for you to save to
 a file.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new CreateKeyPairCommand({
    KeyName: keyName,
  });

  try {
    const { KeyMaterial, KeyName } = await client.send(command);
    console.log(KeyName);
    console.log(KeyMaterial);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidKeyPair.Duplicate") {
      console.warn(`${caught.message}. Try another key name.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateKeyPairà la section Référence des AWS SDK pour JavaScript API.

`CreateLaunchTemplate`

L'exemple de code suivant montre comment utiliser`CreateLaunchTemplate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const ssmClient = new SSMClient({});
const { Parameter } = await ssmClient.send(
  new GetParameterCommand({
    Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
  }),
);
const ec2Client = new EC2Client({});
await ec2Client.send(
  new CreateLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
      UserData: readFileSync(
        join(RESOURCES_PATH, "server_startup_script.sh"),
      ).toString("base64"),
      KeyName: NAMES.keyPairName,
    },
  }),
```

- Pour plus de détails sur l'API, reportez-vous CreateLaunchTemplateà la section Référence des AWS SDK pour JavaScript API.

`CreateSecurityGroup`

L'exemple de code suivant montre comment utiliser`CreateSecurityGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates a security group.
 * @param {{ groupName: string, description: string }} options
 */
export const main = async ({ groupName, description }) => {
  const client = new EC2Client({});
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: groupName,
    // Up to 255 characters in length.
    Description: description,
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateSecurityGroupà la section Référence des AWS SDK pour JavaScript API.

`DeleteKeyPair`

L'exemple de code suivant montre comment utiliser`DeleteKeyPair`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes the specified key pair, by removing the public key from Amazon EC2.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new DeleteKeyPairCommand({
    KeyName: keyName,
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide the required value?`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteKeyPairà la section Référence des AWS SDK pour JavaScript API.

`DeleteLaunchTemplate`

L'exemple de code suivant montre comment utiliser`DeleteLaunchTemplate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- Pour plus de détails sur l'API, reportez-vous DeleteLaunchTemplateà la section Référence des AWS SDK pour JavaScript API.

`DeleteSecurityGroup`

L'exemple de code suivant montre comment utiliser`DeleteSecurityGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes a security group.
 * @param {{ groupId: string }} options
 */
```

```
export const main = async ({ groupId }) => {
  const client = new EC2Client({});
  const command = new DeleteSecurityGroupCommand({
    GroupId: groupId,
  });

  try {
    await client.send(command);
    console.log("Security group deleted successfully.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteSecurityGroupà la section Référence des AWS SDK pour JavaScript API.

`DescribeAddresses`

L'exemple de code suivant montre comment utiliser`DescribeAddresses`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeAddressesCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Describes the specified Elastic IP addresses or all of your Elastic IP addresses.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
```

```
  const client = new EC2Client({});
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: [allocationId],
  });

  try {
    const { Addresses } = await client.send(command);
    const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
    console.log("Elastic IP addresses:");
    console.log(addressList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationId.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeAddressesà la section Référence des AWS SDK pour JavaScript API.

`DescribeIamInstanceProfileAssociations`

L'exemple de code suivant montre comment utiliser`DescribeIamInstanceProfileAssociations`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    const ec2Client = new EC2Client({});
```

```
    const { IamInstanceProfileAssociations } = await ec2Client.send(
      new DescribeIamInstanceProfileAssociationsCommand({
        Filters: [
          { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
        ],
      }),
    );
```

- Pour plus de détails sur l'API, reportez-vous DescribeIamInstanceProfileAssociationsà la section Référence des AWS SDK pour JavaScript API.

`DescribeImages`

L'exemple de code suivant montre comment utiliser`DescribeImages`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, paginateDescribeImages } from "@aws-sdk/client-ec2";

/**
 * Describes the specified images (AMIs, AKIs, and ARIs) available to you or all of
 the images available to you.
 * @param {{ architecture: string, pageSize: number }} options
 */
export const main = async ({ architecture, pageSize }) => {
  pageSize = Number.parseInt(pageSize);
  const client = new EC2Client({});

  // The paginate function is a wrapper around the base command.
  const paginator = paginateDescribeImages(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the base command.
    { client, pageSize },
```

```
    {
      // There are almost 70,000 images available. Be specific with your filtering
      // to increase efficiency.
      // See https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
ec2/interfaces/describeimagescommandinput.html#filters
      Filters: [{ Name: "architecture", Values: [architecture] }],
    },
  );

  /**
   * @type {import('@aws-sdk/client-ec2').Image[]}
   */
  const images = [];
  let recordsScanned = 0;

  try {
    for await (const page of paginator) {
      recordsScanned += pageSize;
      if (page.Images.length) {
        images.push(...page.Images);
        break;
      }
      console.log(
        `No matching image found yet. Searched ${recordsScanned} records.`,
      );
    }

    if (images.length) {
      console.log(
        `Found ${images.length} images:\n\n${images.map((image) =>
 image.Name).join("\n")}\n`,
      );
    } else {
      console.log(
        `No matching images found. Searched ${recordsScanned} records.\n`,
      );
    }

    return images;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
      return [];
    }
```

```
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeImagesà la section Référence des AWS SDK pour JavaScript API.

`DescribeInstanceTypes`

L'exemple de code suivant montre comment utiliser`DescribeInstanceTypes`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, paginateDescribeInstanceTypes } from "@aws-sdk/client-ec2";

/**
 * Describes the specified instance types. By default, all instance types for the
 * current Region are described. Alternatively, you can filter the results.
 * @param {{ pageSize: string, supportedArch: string[], freeTier: boolean }} options
 */
export const main = async ({ pageSize, supportedArch, freeTier }) => {
  pageSize = Number.parseInt(pageSize);
  const client = new EC2Client({});

  // The paginate function is a wrapper around the underlying command.
  const paginator = paginateDescribeInstanceTypes(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the underlying command.
    { client, pageSize },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
```

```
          Values: supportedArch,
        },
        { Name: "free-tier-eligible", Values: [freeTier ? "true" : "false"] },
      ],
    },
  );

  try {
    /**
     * @type {import('@aws-sdk/client-ec2').InstanceTypeInfo[]}
     */
    const instanceTypes = [];

    for await (const page of paginator) {
      if (page.InstanceTypes.length) {
        instanceTypes.push(...page.InstanceTypes);

        // When we have at least 1 result, we can stop.
        if (instanceTypes.length >= 1) {
          break;
        }
      }
    }
    console.log(
      `Memory size in MiB for matching instance types:\n\n${instanceTypes.map((it)
 => `${it.InstanceType}: ${it.MemoryInfo.SizeInMiB} MiB`).join("\n")}`,
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
      return [];
    }
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeInstanceTypesà la section Référence des AWS SDK pour JavaScript API.

`DescribeInstances`

L'exemple de code suivant montre comment utiliser`DescribeInstances`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, paginateDescribeInstances } from "@aws-sdk/client-ec2";

/**
 * List all of your EC2 instances running with the provided architecture that
 * were launched in the past month.
 * @param {{ pageSize: string, architectures: string[] }} options
 */
export const main = async ({ pageSize, architectures }) => {
  pageSize = Number.parseInt(pageSize);
  const client = new EC2Client({});
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;

  const paginator = paginateDescribeInstances(
    {
      client,
      pageSize,
    },
    {
      Filters: [
        { Name: "architecture", Values: architectures },
        { Name: "instance-state-name", Values: ["running"] },
        {
          Name: "launch-time",
          Values: [launchTimePattern],
        },
      ],
    },
  );

  try {
    /**
```

```
     * @type {import('@aws-sdk/client-ec2').Instance[]}
     */
    const instanceList = [];
    for await (const page of paginator) {
      const { Reservations } = page;
      for (const reservation of Reservations) {
        instanceList.push(...reservation.Instances);
      }
    }
    console.log(
      `Running instances launched this month:\n\n${instanceList.map((instance) =>
 instance.InstanceId).join("\n")}`,
    );
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeInstancesà la section Référence des AWS SDK pour JavaScript API.

`DescribeKeyPairs`

L'exemple de code suivant montre comment utiliser`DescribeKeyPairs`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeKeyPairsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
```

```
 * List all key pairs in the current AWS account.
 * @param {{ dryRun: boolean }}
 */
export const main = async ({ dryRun }) => {
  const client = new EC2Client({});
  const command = new DescribeKeyPairsCommand({ DryRun: dryRun });

  try {
    const { KeyPairs } = await client.send(command);
    const keyPairList = KeyPairs.map(
      (kp) => ` • ${kp.KeyPairId}: ${kp.KeyName}`,
    ).join("\n");
    console.log("The following key pairs were found in your account:");
    console.log(keyPairList);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
      console.log(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeKeyPairsà la section Référence des AWS SDK pour JavaScript API.

`DescribeRegions`

L'exemple de code suivant montre comment utiliser`DescribeRegions`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeRegionsCommand, EC2Client } from "@aws-sdk/client-ec2";
```

```
/**
 * List all available AWS regions.
 * @param {{ regionNames: string[], includeOptInRegions: boolean }} options
 */
export const main = async ({ regionNames, includeOptInRegions }) => {
  const client = new EC2Client({});
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions is true, even the regions that require opt-in will be
 returned.
    AllRegions: includeOptInRegions,
    // You can omit the Filters property if you want to get all regions.
    Filters: regionNames?.length
      ? [
          {
            Name: "region-name",
            // You can specify multiple values for a filter.
            // You can also use '*' as a wildcard. This will return all
            // of the regions that start with `us-east-`.
            Values: regionNames,
          },
        ]
      : undefined,
  });

  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
      console.log(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeRegionsà la section Référence des AWS SDK pour JavaScript API.

`DescribeSecurityGroups`

L'exemple de code suivant montre comment utiliser`DescribeSecurityGroups`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Describes the specified security groups or all of your security groups.
 * @param {{ groupIds: string[] }} options
 */
export const main = async ({ groupIds = [] }) => {
  const client = new EC2Client({});
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: groupIds,
  });

  try {
    const { SecurityGroups } = await client.send(command);
    const sgList = SecurityGroups.map(
      (sg) => `• ${sg.GroupName} (${sg.GroupId}): ${sg.Description}`,
    ).join("\n");
    if (sgList.length) {
      console.log(`Security groups:\n${sgList}`);
    } else {
      console.log("No security groups found.");
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else if (
      caught instanceof Error &&
      caught.name === "InvalidGroup.NotFound"
    ) {
      console.warn(caught.message);
    } else {
```

```
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeSecurityGroupsà la section Référence des AWS SDK pour JavaScript API.

`DescribeSubnets`

L'exemple de code suivant montre comment utiliser`DescribeSubnets`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  }),
);
```

- Pour plus de détails sur l'API, reportez-vous DescribeSubnetsà la section Référence des AWS SDK pour JavaScript API.

`DescribeVpcs`

L'exemple de code suivant montre comment utiliser`DescribeVpcs`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
```

- Pour plus de détails sur l'API, reportez-vous DescribeVpcsà la section Référence des AWS SDK pour JavaScript API.

`DisassociateAddress`

L'exemple de code suivant montre comment utiliser`DisassociateAddress`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DisassociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Disassociate an Elastic IP address from an instance.
 * @param {{ associationId: string }} options
 */
export const main = async ({ associationId }) => {
  const client = new EC2Client({});
  const command = new DisassociateAddressCommand({
```

```
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AssociationId: associationId,
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAssociationID.NotFound"
    ) {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DisassociateAddressà la section Référence des AWS SDK pour JavaScript API.

`MonitorInstances`

L'exemple de code suivant montre comment utiliser`MonitorInstances`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, MonitorInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Turn on detailed monitoring for the selected instance.
 * By default, metrics are sent to Amazon CloudWatch every 5 minutes.
```

```
 * For a cost you can enable detailed monitoring which sends metrics every minute.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new MonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instancesBeingMonitored = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instancesBeingMonitored.join("\n"));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous MonitorInstancesà la section Référence des AWS SDK pour JavaScript API.

`RebootInstances`

L'exemple de code suivant montre comment utiliser`RebootInstances`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, RebootInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Requests a reboot of the specified instances. This operation is asynchronous;
 * it only queues a request to reboot the specified instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new RebootInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Please provide the InstanceId of a valid instance to
 reboot.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous RebootInstancesà la section Référence des AWS SDK pour JavaScript API.

`ReleaseAddress`

L'exemple de code suivant montre comment utiliser`ReleaseAddress`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { ReleaseAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Release an Elastic IP address.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
  const client = new EC2Client({});
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AllocationId: allocationId,
  });

  try {
    await client.send(command);
    console.log("Successfully released address.");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationID.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ReleaseAddressà la section Référence des AWS SDK pour JavaScript API.

`ReplaceIamInstanceProfileAssociation`

L'exemple de code suivant montre comment utiliser`ReplaceIamInstanceProfileAssociation`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );
```

- Pour plus de détails sur l'API, reportez-vous ReplaceIamInstanceProfileAssociationà la section Référence des AWS SDK pour JavaScript API.

`RunInstances`

L'exemple de code suivant montre comment utiliser`RunInstances`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, RunInstancesCommand } from "@aws-sdk/client-ec2";

/**
```

```
 * Create new EC2 instances.
 * @param {{
 *  keyName: string,
 *  securityGroupIds: string[],
 *  imageId: string,
 *  instanceType: import('@aws-sdk/client-ec2')._InstanceType,
 *  minCount?: number,
 *  maxCount?: number }} options
 */
export const main = async ({
  keyName,
  securityGroupIds,
  imageId,
  instanceType,
  minCount = "1",
  maxCount = "1",
}) => {
  const client = new EC2Client({});
  minCount = Number.parseInt(minCount);
  maxCount = Number.parseInt(maxCount);
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: keyName,
    // Your security group.
    SecurityGroupIds: securityGroupIds,
    // An Amazon Machine Image (AMI). There are multiple ways to search for AMIs.
 For more information, see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html
    ImageId: imageId,
    // An instance type describing the resources provided to your instance. There
 are multiple
    // ways to search for instance types. For more information see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-discovery.html
    InstanceType: instanceType,
    // Availability Zones have capacity limitations that may impact your ability to
 launch instances.
    // The `RunInstances` operation will only succeed if it can allocate at least
 the `MinCount` of instances.
    // However, EC2 will attempt to launch up to the `MaxCount` of instances, even
 if the full request cannot be satisfied.
    // If you need a specific number of instances, use `MinCount` and `MaxCount` set
 to the same value.
    // If you want to launch up to a certain number of instances, use `MaxCount` and
 let EC2 provision as many as possible.
```

```
    // If you require a minimum number of instances, but do not want to exceed a
 maximum, use both `MinCount` and `MaxCount`.
    MinCount: minCount,
    MaxCount: maxCount,
  });

  try {
    const { Instances } = await client.send(command);
    const instanceList = Instances.map(
      (instance) => `• ${instance.InstanceId}`,
    ).join("\n");
    console.log(`Launched instances:\n${instanceList}`);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceCountExceeded") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous RunInstancesà la section Référence des AWS SDK pour JavaScript API.

`StartInstances`

L'exemple de code suivant montre comment utiliser`StartInstances`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, StartInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";
```

```
/**
 * Starts an Amazon EBS-backed instance that you've previously stopped.
 * @param {{ instanceIds }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StartInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous StartInstancesà la section Référence des AWS SDK pour JavaScript API.

`StopInstances`

L'exemple de code suivant montre comment utiliser`StopInstances`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, StopInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Stop one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StopInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous StopInstancesà la section Référence des AWS SDK pour JavaScript API.

`TerminateInstances`

L'exemple de code suivant montre comment utiliser`TerminateInstances`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, TerminateInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Terminate one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new TerminateInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
```

```
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous TerminateInstancesà la section Référence des AWS SDK pour JavaScript API.

`UnmonitorInstances`

L'exemple de code suivant montre comment utiliser`UnmonitorInstances`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { EC2Client, UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Turn off detailed monitoring for the selected instance.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new UnmonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instanceMonitoringsList = InstanceMonitorings.map(
      (im) =>
```

```
        ` • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instanceMonitoringsList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous UnmonitorInstancesà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer et gérer un service résilient

L'exemple de code suivant montre comment créer un service Web à charge équilibrée qui renvoie des recommandations de livres, de films et de chansons. L'exemple montre comment le service répond aux défaillances et comment le restructurer pour accroître la résilience en cas de défaillance.

- Utilisez un groupe Amazon EC2 Auto Scaling pour créer des instances Amazon Elastic Compute Cloud (Amazon EC2) sur la base d'un modèle de lancement et pour maintenir le nombre d'instances dans une plage spécifiée.
- Gérez et distribuez les requêtes HTTP avec Elastic Load Balancing.
- Surveillez l’état des instances d’un groupe Auto Scaling et transférez les demandes uniquement aux instances saines.
- Exécutez un serveur Web Python sur chaque EC2 instance pour gérer les requêtes HTTP. Le serveur Web répond par des recommandations et des surveillances de l’état.
- Simulez un service de recommandation avec une table Amazon DynamoDB.

- Contrôlez la réponse du serveur Web aux demandes et aux contrôles de santé en mettant à jour AWS Systems Manager les paramètres.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

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

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

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

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

/**
```

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

// Call function if run directly
import { fileURLToPath } from "node:url";

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

Créez des étapes pour déployer toutes les ressources.

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
```

```
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
```

```
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
```

```
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );
```

```
    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
```

```
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
```

```
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
```

```
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
```

```
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
```

```
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
```

```
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
```

```
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
```

```
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      }
      return MESSAGES.noIpRules;
    },
```

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

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    }
    return false;
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
```

```
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

Créez des étapes pour exécuter la démo.

```
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;
}
```

Créez des étapes pour déployer toutes les ressources.

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

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
```

```
import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
```

```
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
    return MESSAGES.deletedKeyPair.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

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

```
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
```

```
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
    return MESSAGES.deletedInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
```

```
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
    return MESSAGES.deletedInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
```

```
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
```

```
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
```

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

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

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

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

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

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

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

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

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

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript .
  - AttachLoadBalancerTargetGroups

- CreateAutoScalingGroup
- CreateInstanceProfile
- CreateLaunchTemplate
- CreateListener
- CreateLoadBalancer
- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Elastic Load Balancing - Exemples de version 2 utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Elastic Load Balancing - Version 2.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour Elastic Load Balancing

L'exemple de code suivant montre comment commencer à utiliser Elastic Load Balancing.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
```

```
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeLoadBalancersà la section Référence des AWS SDK pour JavaScript API.

Actions

`CreateListener`

L'exemple de code suivant montre comment utiliser`CreateListener`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
```

```
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  );
```

- Pour plus de détails sur l'API, reportez-vous CreateListeneràla section Référence des AWS SDK pour JavaScript API.

`CreateLoadBalancer`

L'exemple de code suivant montre comment utiliser`CreateLoadBalancer`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new CreateLoadBalancerCommand({
    Name: NAMES.loadBalancerName,
    Subnets: state.subnets,
  }),
);
state.loadBalancerDns = LoadBalancers[0].DNSName;
state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
await waitUntilLoadBalancerAvailable(
  { client },
  { Names: [NAMES.loadBalancerName] },
);
```

- Pour plus de détails sur l'API, reportez-vous CreateLoadBalancerà la section Référence des AWS SDK pour JavaScript API.

`CreateTargetGroup`

L'exemple de code suivant montre comment utiliser`CreateTargetGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new CreateTargetGroupCommand({
    Name: NAMES.loadBalancerTargetGroupName,
    Protocol: "HTTP",
    Port: 80,
    HealthCheckPath: "/healthcheck",
    HealthCheckIntervalSeconds: 10,
    HealthCheckTimeoutSeconds: 5,
    HealthyThresholdCount: 2,
    UnhealthyThresholdCount: 2,
    VpcId: state.defaultVpc,
  }),
);
```

- Pour plus de détails sur l'API, reportez-vous CreateTargetGroupà la section Référence des AWS SDK pour JavaScript API.

`DeleteLoadBalancer`

L'exemple de code suivant montre comment utiliser`DeleteLoadBalancer`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const client = new ElasticLoadBalancingV2Client({});
const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
await client.send(
  new DeleteLoadBalancerCommand({
    LoadBalancerArn: loadBalancer.LoadBalancerArn,
  }),
);
await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
  const lb = await findLoadBalancer(NAMES.loadBalancerName);
  if (lb) {
    throw new Error("Load balancer still exists.");
  }
});
```

- Pour plus de détails sur l'API, reportez-vous DeleteLoadBalancerà la section Référence des AWS SDK pour JavaScript API.

`DeleteTargetGroup`

L'exemple de code suivant montre comment utiliser`DeleteTargetGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const client = new ElasticLoadBalancingV2Client({});
```

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

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteTargetGroupà la section Référence des AWS SDK pour JavaScript API.

`DescribeLoadBalancers`

L'exemple de code suivant montre comment utiliser`DescribeLoadBalancers`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
```

```
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeLoadBalancersà la section Référence des AWS SDK pour JavaScript API.

`DescribeTargetGroups`

L'exemple de code suivant montre comment utiliser`DescribeTargetGroups`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

- Pour plus de détails sur l'API, reportez-vous DescribeTargetGroupsà la section Référence des AWS SDK pour JavaScript API.

`DescribeTargetHealth`

L'exemple de code suivant montre comment utiliser`DescribeTargetHealth`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

- Pour plus de détails sur l'API, reportez-vous DescribeTargetHealthà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer et gérer un service résilient

L'exemple de code suivant montre comment créer un service Web à charge équilibrée qui renvoie des recommandations de livres, de films et de chansons. L'exemple montre comment le service répond aux défaillances et comment le restructurer pour accroître la résilience en cas de défaillance.

- Utilisez un groupe Amazon EC2 Auto Scaling pour créer des instances Amazon Elastic Compute Cloud (Amazon EC2) sur la base d'un modèle de lancement et pour maintenir le nombre d'instances dans une plage spécifiée.
- Gérez et distribuez les requêtes HTTP avec Elastic Load Balancing.

- Surveillez l'état des instances d'un groupe Auto Scaling et transférez les demandes uniquement aux instances saines.
- Exécutez un serveur Web Python sur chaque EC2 instance pour gérer les requêtes HTTP. Le serveur Web répond par des recommandations et des surveillances de l'état.
- Simulez un service de recommandation avec une table Amazon DynamoDB.
- Contrôlez la réponse du serveur Web aux demandes et aux contrôles de santé en mettant à jour AWS Systems Manager les paramètres.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

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

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

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

/**
```

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

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

// Call function if run directly
import { fileURLToPath } from "node:url";

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

Créez des étapes pour déployer toutes les ressources.

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
```

```
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
```

```
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
```

```
          KeyType: "RANGE",
        },
      ],
    }),
  );
  await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
  "createdTable",
  MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "populatingTable",
  MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
```

```
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
```

```
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
```

```
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
```

```
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
```

```
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
```

```
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
```

```
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
  state.targetGroupProtocol = targetGroup.Protocol;
  state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
  "createdLoadBalancerTargetGroup",
  MESSAGES.createdLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
    }),
  );
  state.loadBalancerDns = LoadBalancers[0].DNSName;
  state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
  await waitUntilLoadBalancerAvailable(
    { client },
    { Names: [NAMES.loadBalancerName] },
  );
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
  MESSAGES.createdLoadBalancer
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioOutput(
  "creatingListener",
  MESSAGES.creatingLoadBalancerListener
    .replace("${LB_NAME}", NAMES.loadBalancerName)
```

```
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
),
new ScenarioAction("createListener", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { Listeners } = await client.send(
    new CreateListenerCommand({
      LoadBalancerArn: state.loadBalancerArn,
      Protocol: state.targetGroupProtocol,
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  );
  const listener = Listeners[0];
  state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
  MESSAGES.createdLoadBalancerListener.replace(
    "${LB_LISTENER_ARN}",
    state.loadBalancerListenerArn,
  ),
),
new ScenarioOutput(
  "attachingLoadBalancerTargetGroup",
  MESSAGES.attachingLoadBalancerTargetGroup
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
    .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
),
new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
}),
new ScenarioOutput(
  "attachedLoadBalancerTargetGroup",
  MESSAGES.attachedLoadBalancerTargetGroup,
),
new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
new ScenarioAction(
```

```
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
```

```
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      }
      return MESSAGES.noIpRules;
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      return MESSAGES.noIpRules;
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
```

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

Créez des étapes pour exécuter la démo.

```
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;
}
```

Créez des étapes pour déployer toutes les ressources.

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

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
```

```
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
```

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

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
```

```
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
```

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

```
      NAMES.instanceRoleName,
    );
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
    return MESSAGES.deletedInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
```

```
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
```

```
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
```

```
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
```

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

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

```
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
```

```
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
  }),
];

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

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

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

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

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Résolution des entités AWS exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec Résolution des entités AWS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour Résolution des entités AWS

L'exemple de code suivant montre comment démarrer avec Résolution des entités AWS.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  EntityResolutionClient,
  ListMatchingWorkflowsCommand,
} from "@aws-sdk/client-entityresolution";

export const main = async () => {
  const region = "eu-west-1";
```

```
  const erClient = new EntityResolutionClient({ region: region });
  try {
    const command = new ListMatchingWorkflowsCommand({});
    const response = await erClient.send(command);
    const workflowSummaries = response.workflowSummaries;
    for (const workflowSummary of workflowSummaries) {
      console.log(`Attribute name: ${workflowSummaries[0].workflowName} `);
    }
    if (workflowSummaries.length === 0) {
      console.log("No matching workflows found.");
    }
  } catch (error) {
    console.error(
      `An error occurred in listing the workflow summaries: ${error.message} \n
 Exiting program.`,
    );
    return;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ListMatchingWorkflowsà la section Référence des AWS SDK pour JavaScript API.

Actions

`CreateMatchingWorkflow`

L'exemple de code suivant montre comment utiliser`CreateMatchingWorkflow`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.
```

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

import {
  CreateMatchingWorkflowCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const createMatchingWorkflowParams = {
    roleArn: `${data.inputs.roleArn}`,
    workflowName: `${data.inputs.workflowName}`,
    description: "Created by using the AWS SDK for JavaScript (v3).",
    inputSourceConfig: [
      {
        inputSourceARN: `${data.inputs.JSONinputSourceARN}`,
        schemaName: `${data.inputs.schemaNameJson}`,
        applyNormalization: false,
      },
      {
        inputSourceARN: `${data.inputs.CSVinputSourceARN}`,
        schemaName: `${data.inputs.schemaNameCSV}`,
        applyNormalization: false,
      },
    ],
    outputSourceConfig: [
      {
        outputS3Path: `s3://${data.inputs.myBucketName}/eroutput`,
        output: [
          {
            name: "id",
          },
          {
            name: "name",
          },
          {
            name: "email",
          },
          {
            name: "phone",
```

```
          },
        ],
        applyNormalization: false,
      },
    ],
    resolutionTechniques: { resolutionType: "ML_MATCHING" },
  };
  try {
    const command = new CreateMatchingWorkflowCommand(
      createMatchingWorkflowParams,
    );
    const response = await erClient.send(command);

    console.log(
      `Workflow created successfully.\n The workflow ARN is:
 ${response.workflowArn}`,
    );
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateMatchingWorkflowà la section Référence des AWS SDK pour JavaScript API.

`CreateSchemaMapping`

L'exemple de code suivant montre comment utiliser`CreateSchemaMapping`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  CreateSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const createSchemaMappingParamsJson = {
    schemaName: `${data.inputs.schemaNameJson}`,
    mappedInputFields: [
      {
        fieldName: "id",
        type: "UNIQUE_ID",
      },
      {
        fieldName: "name",
        type: "NAME",
      },
      {
        fieldName: "email",
        type: "EMAIL_ADDRESS",
      },
    ],
  };
  const createSchemaMappingParamsCSV = {
    schemaName: `${data.inputs.schemaNameCSV}`,
    mappedInputFields: [
      {
        fieldName: "id",
        type: "UNIQUE_ID",
      },
      {
        fieldName: "name",
        type: "NAME",
      },
      {
        fieldName: "email",
```

```
        type: "EMAIL_ADDRESS",
      },
      {
        fieldName: "phone",
        type: "PROVIDER_ID",
        subType: "STRING",
      },
    ],
  };
  try {
    const command = new CreateSchemaMappingCommand(
      createSchemaMappingParamsJson,
    );
    const response = await erClient.send(command);
    console.log("The JSON schema mapping name is ", response.schemaName);
  } catch (error) {
    console.log("error ", error.message);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateSchemaMappingà la section Référence des AWS SDK pour JavaScript API.

`DeleteMatchingWorkflow`

L'exemple de code suivant montre comment utiliser`DeleteMatchingWorkflow`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.

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

```
import {
  DeleteMatchingWorkflowCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  try {
    const deleteWorkflowParams = {
      workflowName: `${data.inputs.workflowName}`,
    };
    const command = new DeleteMatchingWorkflowCommand(deleteWorkflowParams);
    const response = await erClient.send(command);
    console.log("Workflow deleted successfully!", response);
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteMatchingWorkflowà la section Référence des AWS SDK pour JavaScript API.

`DeleteSchemaMapping`

L'exemple de code suivant montre comment utiliser`DeleteSchemaMapping`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  DeleteSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const deleteSchemaMapping = {
    schemaName: `${data.inputs.schemaNameJson}`,
  };
  try {
    const command = new DeleteSchemaMappingCommand(deleteSchemaMapping);
    const response = await erClient.send(command);
    console.log("Schema mapping deleted successfully. ", response);
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteSchemaMappingà la section Référence des AWS SDK pour JavaScript API.

`GetMatchingJob`

L'exemple de code suivant montre comment utiliser`GetMatchingJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  GetMatchingJobCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  async function getInfo() {
    const getJobInfoParams = {
      workflowName: `${data.inputs.workflowName}`,
      jobId: `${data.inputs.jobId}`,
    };
    try {
      const command = new GetMatchingJobCommand(getJobInfoParams);
      const response = await erClient.send(command);
      console.log(`Job status: ${response.status}`);
    } catch (error) {
      console.log("error ", error.message);
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous GetMatchingJobà la section Référence des AWS SDK pour JavaScript API.

`GetSchemaMapping`

L'exemple de code suivant montre comment utiliser`GetSchemaMapping`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  GetSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const getSchemaMappingJsonParams = {
    schemaName: `${data.inputs.schemaNameJson}`,
  };
  try {
    const command = new GetSchemaMappingCommand(getSchemaMappingJsonParams);
    const response = await erClient.send(command);
    console.log(response);
    console.log(
      `Schema mapping for the JSON data:\n ${response.mappedInputFields[0]}`,
    );
    console.log("Schema mapping ARN is: ", response.schemaArn);
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous GetSchemaMappingà la section Référence des AWS SDK pour JavaScript API.

`ListSchemaMappings`

L'exemple de code suivant montre comment utiliser`ListSchemaMappings`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  ListSchemaMappingsCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  async function getInfo() {
    const listSchemaMappingsParams = {
      workflowName: `${data.inputs.workflowName}`,
      jobId: `${data.inputs.jobId}`,
    };
    try {
      const command = new ListSchemaMappingsCommand(listSchemaMappingsParams);
      const response = await erClient.send(command);
      const noOfSchemas = response.schemaList.length;
      for (let i = 0; i < noOfSchemas; i++) {
        console.log(
          `Schema Mapping Name: ${response.schemaList[i].schemaName} `,
```

```
        );
      }
    } catch (caught) {
      console.error(caught.message);
      throw caught;
    }
  }
```

- Pour plus de détails sur l'API, reportez-vous ListSchemaMappingsà la section Référence des AWS SDK pour JavaScript API.

`StartMatchingJob`

L'exemple de code suivant montre comment utiliser`StartMatchingJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";
import {
  StartMatchingJobCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const matchingJobOfWorkflowParams = {
    workflowName: `${data.inputs.workflowName}`,
```

```
  };
  try {
    const command = new StartMatchingJobCommand(matchingJobOfWorkflowParams);
    const response = await erClient.send(command);
    console.log(`Job ID: ${response.jobID} \n
The matching job was successfully started.`);
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous StartMatchingJobà la section Référence des AWS SDK pour JavaScript API.

`TagResource`

L'exemple de code suivant montre comment utiliser`TagResource`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  TagResourceCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
```

```
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const tagResourceCommandParams = {
    resourceArn: `${data.inputs.schemaArn}`,
    tags: {
      tag1: "tag1Value",
      tag2: "tag2Value",
    },
  };
  try {
    const command = new TagResourceCommand(tagResourceCommandParams);
    const response = await erClient.send(command);
    console.log("Successfully tagged the resource.");
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous TagResourceà la section Référence des AWS SDK pour JavaScript API.

EventBridge exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec EventBridge.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d’un même service ou combinés à d’autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l’exécution du code en contexte.

Rubriques

Actions

`PutEvents`

L'exemple de code suivant montre comment utiliser`PutEvents`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import {
  EventBridgeClient,
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";

export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutEventsCommand({
      Entries: [
        {
          Detail: JSON.stringify({ greeting: "Hello there." }),
          DetailType: detailType,
          Resources: resources,
          Source: source,
        },
```

```
      ],
    }),
  );

  console.log("PutEvents response:");
  console.log(response);
  // PutEvents response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous PutEventsà la section Référence des AWS SDK pour JavaScript API.

`PutRule`

L'exemple de code suivant montre comment utiliser`PutRule`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";
```

```
export const putRule = async (
  ruleName = "some-rule",
  source = "some-source",
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutRuleCommand({
      Name: ruleName,
      EventPattern: JSON.stringify({ source: [source] }),
      State: "ENABLED",
      EventBusName: "default",
    }),
  );

  console.log("PutRule response:");
  console.log(response);
  // PutRule response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/
EventBridgeTestRule-1696280037720'
  // }
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous PutRuleà la section Référence des AWS SDK pour JavaScript API.

`PutTargets`

L'exemple de code suivant montre comment utiliser`PutTargets`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Importez le kit SDK et les modules client et appelez l'API.

```
import {
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   FailedEntries: [],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous PutTargetsà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Utilisent des événements planifiés pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par un événement EventBridge planifié par Amazon.

SDK pour JavaScript (v3)

Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- DynamoDB
- EventBridge

- Lambda
- Amazon SNS

Exemples d'Amazon Glacier utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Glacier.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`CreateVault`

L'exemple de code suivant montre comment utiliser`CreateVault`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
```

```
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Créez le coffre.

```
// 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();
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateVaultà la section Référence des AWS SDK pour JavaScript API.

`UploadArchive`

L'exemple de code suivant montre comment utiliser`UploadArchive`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Chargez l'archive.

```
// 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();
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous UploadArchiveà la section Référence des AWS SDK pour JavaScript API.

AWS Glue exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec AWS Glue.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour AWS Glue

L'exemple de code suivant montre comment démarrer avec AWS Glue.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

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

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

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- Pour plus de détails sur l'API, reportez-vous ListJobsà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez un Crawler qui indexe un compartiment Amazon S3 public et génère une base de données de métadonnées au format CSV.
- Répertoriez les informations relatives aux bases de données et aux tables de votre AWS Glue Data Catalog.
- Créez une tâche pour extraire les données CSV du compartiment S3, transformer les données et charger la sortie au format JSON dans un autre compartiment S3.
- Répertoriez les informations relatives aux exécutions de tâches, visualisez les données transformées et nettoyez les ressources.

Pour plus d'informations, consultez Tutoriel : prise en main de AWS Glue Studio.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez et exécutez un crawler qui analyse un compartiment Amazon Simple Storage Service (Amazon S3) public et génère une base de données de métadonnées qui décrit les données au format CSV trouvées.

```
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 };
  };
```

Répertoriez les informations relatives aux bases de données et aux tables de votre 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 };
  };
```

Créez et exécutez une tâche qui extrait les données CSV du compartiment Amazon S3 source, les transforme en supprimant et en renommant des champs, et charge la sortie au format JSON dans un autre compartiment Amazon S3.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });
```

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

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput> }  getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
```

```
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }

  switch (JobRun.JobRunState) {
    case "FAILED":
    case "TIMEOUT":
    case "STOPPED":
    case "ERROR":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "SUCCEEDED":
      return;
    default:
      break;
  }

  log(
    `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
  );
  await wait(waitTimeInSeconds);
  return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
 view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
```

```
  async (context) => {
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);

    return { ...context };
  };
```

Répertoriez les informations relatives aux exécutions de tâches et affichez certaines des données transformées.

```
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 };
  };
```

Supprimez toutes les ressources créées par la démonstration.

```
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 };
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler

- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Actions

`CreateCrawler`

L'exemple de code suivant montre comment utiliser`CreateCrawler`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });
```

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

- Pour plus de détails sur l'API, reportez-vous CreateCrawlerà la section Référence des AWS SDK pour JavaScript API.

`CreateJob`

L'exemple de code suivant montre comment utiliser`CreateJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

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

- Pour plus de détails sur l'API, reportez-vous CreateJobà la section Référence des AWS SDK pour JavaScript API.

`DeleteCrawler`

L'exemple de code suivant montre comment utiliser`DeleteCrawler`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

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

- Pour plus de détails sur l'API, reportez-vous DeleteCrawlerà la section Référence des AWS SDK pour JavaScript API.

`DeleteDatabase`

L'exemple de code suivant montre comment utiliser`DeleteDatabase`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const deleteDatabase = (databaseName) => {
```

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

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

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

- Pour plus de détails sur l'API, reportez-vous DeleteDatabaseà la section Référence des AWS SDK pour JavaScript API.

`DeleteJob`

L'exemple de code suivant montre comment utiliser`DeleteJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

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

- Pour plus de détails sur l'API, reportez-vous DeleteJobà la section Référence des AWS SDK pour JavaScript API.

`DeleteTable`

L'exemple de code suivant montre comment utiliser`DeleteTable`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

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

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour JavaScript API.

`GetCrawler`

L'exemple de code suivant montre comment utiliser`GetCrawler`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getCrawler = (name) => {
```

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

  const command = new GetCrawlerCommand({
    Name: name,
  });

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

- Pour plus de détails sur l'API, reportez-vous GetCrawlerà la section Référence des AWS SDK pour JavaScript API.

`GetDatabase`

L'exemple de code suivant montre comment utiliser`GetDatabase`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

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

- Pour plus de détails sur l'API, reportez-vous GetDatabaseà la section Référence des AWS SDK pour JavaScript API.

`GetDatabases`

L'exemple de code suivant montre comment utiliser`GetDatabases`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

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

- Pour plus de détails sur l'API, reportez-vous GetDatabasesà la section Référence des AWS SDK pour JavaScript API.

`GetJob`

L'exemple de code suivant montre comment utiliser`GetJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getJob = (jobName) => {
  const client = new GlueClient({});
```

```
  const command = new GetJobCommand({
    JobName: jobName,
  });

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

- Pour plus de détails sur l'API, reportez-vous GetJobà la section Référence des AWS SDK pour JavaScript API.

`GetJobRun`

L'exemple de code suivant montre comment utiliser`GetJobRun`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

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

- Pour plus de détails sur l'API, reportez-vous GetJobRunà la section Référence des AWS SDK pour JavaScript API.

`GetJobRuns`

L'exemple de code suivant montre comment utiliser`GetJobRuns`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

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

- Pour plus de détails sur l'API, reportez-vous GetJobRunsà la section Référence des AWS SDK pour JavaScript API.

`GetTables`

L'exemple de code suivant montre comment utiliser`GetTables`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });
```

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

- Pour plus de détails sur l'API, reportez-vous GetTablesà la section Référence des AWS SDK pour JavaScript API.

`ListJobs`

L'exemple de code suivant montre comment utiliser`ListJobs`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

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

- Pour plus de détails sur l'API, reportez-vous ListJobsà la section Référence des AWS SDK pour JavaScript API.

`StartCrawler`

L'exemple de code suivant montre comment utiliser`StartCrawler`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

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

- Pour plus de détails sur l'API, reportez-vous StartCrawlerà la section Référence des AWS SDK pour JavaScript API.

`StartJobRun`

L'exemple de code suivant montre comment utiliser`StartJobRun`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
```

```
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

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

- Pour plus de détails sur l'API, reportez-vous StartJobRunà la section Référence des AWS SDK pour JavaScript API.

HealthImaging exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec HealthImaging.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Actions
- Scénarios

Mise en route

Bonjour HealthImaging

L'exemple de code suivant montre comment démarrer avec HealthImaging.

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Actions

`CopyImageSet`

L'exemple de code suivant montre comment utiliser`CopyImageSet`.

SDK pour JavaScript (v3)

Fonction utilitaire pour copier un ensemble d'images.

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
```

```
/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 image set.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = "",
  force = false,
  copySubsets = [],
) => {
  try {
    const params = {
      datastoreId: datastoreId,
      sourceImageSetId: imageSetId,
      copyImageSetInformation: {
        sourceImageSet: { latestVersionId: sourceVersionId },
      },
      force: force,
    };
    if (destinationImageSetId !== "" && destinationVersionId !== "") {
      params.copyImageSetInformation.destinationImageSet = {
        imageSetId: destinationImageSetId,
        latestVersionId: destinationVersionId,
      };
    }

    if (copySubsets.length > 0) {
      let copySubsetsJson;
      copySubsetsJson = {
        SchemaVersion: 1.1,
        Study: {
          Series: {
            imageSetId: {
```

```
              Instances: {},
            },
          },
        },
      };

      for (let i = 0; i < copySubsets.length; i++) {
        copySubsetsJson.Study.Series.imageSetId.Instances[copySubsets[i]] = {};
      }

      params.copyImageSetInformation.dicomCopies = copySubsetsJson;
    }

    const response = await medicalImagingClient.send(
      new CopyImageSetCommand(params),
    );
    console.log(response);
    // {
    //     '$metadata': {
    //         httpStatusCode: 200,
    //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    //      },
    //       datastoreId: 'xxxxxxxxxxxxx',
    //       destinationImageSetProperties: {
    //             createdAt: 2023-09-27T19:46:21.824Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING',
    //             latestVersionId: '1',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //       },
    //       sourceImageSetProperties: {
    //             createdAt: 2023-09-22T14:49:26.427Z,
    //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxx',
    //             imageSetId: 'xxxxxxxxxxxxxx',
    //             imageSetState: 'LOCKED',
    //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
```

```
    //             latestVersionId: '4',
    //             updatedAt: 2023-09-27T19:46:21.824Z
    //      }
    // }
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

Copiez un ensemble d'images sans destination.

```
  await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
  );
```

Copiez un ensemble d'images avec une destination.

```
  await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
    "12345678901234567890123456789012",
    "1",
    false,
  );
```

Copiez un sous-ensemble d'un ensemble d'images avec une destination et forcez la copie.

```
  await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
```

```
    "12345678901234567890123456789012",
    "1",
    true,
    ["12345678901234567890123456789012", "11223344556677889900112233445566"],
  );
```

- Pour plus de détails sur l'API, reportez-vous CopyImageSetà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`CreateDatastore`

L'exemple de code suivant montre comment utiliser`CreateDatastore`.

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous CreateDatastoreà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`DeleteDatastore`

L'exemple de code suivant montre comment utiliser`DeleteDatastore`.

SDK pour 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;
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteDatastoreà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`DeleteImageSet`

L'exemple de code suivant montre comment utiliser`DeleteImageSet`.

SDK pour 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: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`GetDICOMImportJob`

L'exemple de code suivant montre comment utiliser`GetDICOMImportJob`.

SDK pour JavaScript (v3)

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxx",
```

```
  jobId = "xxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }

  return response;
};
```

- Pour plus de détails sur l'API, voir Get DICOMImport Job in AWS SDK pour JavaScript API Reference.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code AWS.

`GetDatastore`

L'exemple de code suivant montre comment utiliser`GetDatastore`.

SDK pour JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response.datastoreProperties;
};
```

- Pour plus de détails sur l'API, reportez-vous GetDatastoreà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`GetImageFrame`

L'exemple de code suivant montre comment utiliser`GetImageFrame`.

SDK pour JavaScript (v3)

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID",
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    }),
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
```

```
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     contentType: 'application/octet-stream',
  //     imageFrameBlob: <ref *1> IncomingMessage {}
  // }
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous GetImageFrameà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`GetImageSet`

L'exemple de code suivant montre comment utiliser`GetImageSet`.

SDK pour JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imageSetId };
```

```
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`GetImageSetMetadata`

L'exemple de code suivant montre comment utiliser`GetImageSetMetadata`.

SDK pour JavaScript (v3)

Fonction utilitaire pour obtenir les métadonnées d’un ensemble d’images.

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "node:fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxx",
  versionID = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params),
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
```

```
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

Obtenez les métadonnées d'un ensemble d'images sans version.

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
    );
  } catch (err) {
    console.log("Error", err);
  }
```

Obtenez les métadonnées d'un ensemble d'images avec une version.

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1",
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetMetadataà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`ListDICOMImportJobs`

L'exemple de code suivant montre comment utiliser`ListDICOMImportJobs`.

SDK pour JavaScript (v3)

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  const jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page.jobSummaries);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
```

```
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
  // }
  // ]}

  return jobSummaries;
};
```

- Pour plus de détails sur l'API, consultez la section DICOMImportRépertorier les tâches dans la référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`ListDatastores`

L'exemple de code suivant montre comment utiliser`ListDatastores`.

SDK pour JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };
```

```
  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page.datastoreSummaries);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
  //   ]
  // }

  return datastoreSummaries;
};
```

- Pour plus de détails sur l'API, reportez-vous ListDatastoresà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`ListImageSetVersions`

L'exemple de code suivant montre comment utiliser`ListImageSetVersions`.

SDK pour JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams,
  );

  const imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page.imageSetPropertiesList);
    console.log(page);
  }
  // {
  //     '$metadata': {
```

```
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Pour plus de détails sur l'API, reportez-vous ListImageSetVersionsà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`ListTagsForResource`

L'exemple de code suivant montre comment utiliser`ListTagsForResource`.

SDK pour JavaScript (v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
```

```
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi",
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous ListTagsForResourceà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`SearchImageSets`

L'exemple de code suivant montre comment utiliser`SearchImageSets`.

SDK pour JavaScript (v3)

La fonction utilitaire permettant de rechercher des ensembles d'images.

```
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;
};
```

Cas d'utilisation n° 1 : opérateur EQUAL.

```
  const datastoreId = "12345678901234567890123456789012";

  try {
    const searchCriteria = {
      filters: [
        {
          values: [{ DICOMPatientId: "1234567" }],
          operator: "EQUAL",
        },
      ],
    };

    await searchImageSets(datastoreId, searchCriteria);
  } catch (err) {
    console.error(err);
  }
```

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOMStudy la date et l' DICOMStudyheure.

```
  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);
}
```

Cas d'utilisation n° 3 : opérateur BETWEEN utilisant createdAt. Les études temporelles ont été maintenues.

```
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);
}
```

Cas d'utilisation #4 : opérateur DICOMSeries EQUAL sur InstanceUid et BETWEEN sur UpdatedAt et tri la réponse dans l'ordre ASC sur le champ UpdatedAt.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [
      {
        values: [
          { updatedAt: new Date("1985-04-12T23:20:50.52Z") },
          { updatedAt: new Date() },
        ],
        operator: "BETWEEN",
      },
      {
        values: [
          {
            DICOMSeriesInstanceUID:
              "1.1.123.123456.1.12.1.1234567890.1234.12345678.123",
          },
        ],
        operator: "EQUAL",
      },
    ],
    sort: {
      sortOrder: "ASC",
      sortField: "updatedAt",
    },
  };

  await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
  console.error(err);
}
```

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`StartDICOMImportJob`

L'exemple de code suivant montre comment utiliser`StartDICOMImportJob`.

SDK pour JavaScript (v3)

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
 stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/",
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    }),
  );
  console.log(response);
  // {
  //     '$metadata': {
```

```
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- Pour plus de détails sur l'API, consultez Start DICOMImport Job dans le guide de référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`TagResource`

L'exemple de code suivant montre comment utiliser`TagResource`.

SDK pour JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                     - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
```

```
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {},
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous TagResourceà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`UntagResource`

L'exemple de code suivant montre comment utiliser`UntagResource`.

SDK pour JavaScript (v3)

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
```

```
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous UntagResourceà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

`UpdateImageSetMetadata`

L'exemple de code suivant montre comment utiliser`UpdateImageSetMetadata`.

SDK pour JavaScript (v3)

```
import { UpdateImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
      }),
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
```

```
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

Cas d'utilisation n° 1 : insérer ou mettre à jour un attribut et forcer la mise à jour.

```
    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,
    );
```

Cas d'utilisation n° 2 : supprimer un attribut.

```
    // 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,
  );
```

Cas d'utilisation n° 3 : supprimer une instance.

```
  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,
);
```

Cas d'utilisation n° 4 : revenir à une version antérieure.

```
const updateMetadata = {
  revertToVersionId: "1",
};

await updateImageSetMetadata(
  datastoreID,
  imageSetID,
  versionID,
  updateMetadata,
);
```

- Pour plus de détails sur l'API, reportez-vous UpdateImageSetMetadataà la section Référence des AWS SDK pour JavaScript API.

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Scénarios

Mise en route avec les ensembles d'images et les cadres d'images

L'exemple de code suivant montre comment importer des fichiers DICOM et télécharger des cadres d'image dans HealthImaging.

L'implémentation est structurée comme une application de ligne de commande.

- Configurez les ressources pour une importation DICOM.
- Importez des fichiers DICOM dans un magasin de données.
- Récupérez l'ensemble d'images IDs pour la tâche d'importation.

- Récupérez le cadre d'image IDs pour les ensembles d'images.
- Téléchargez, décodez et vérifiez les cadres d’image.
- nettoyer les ressources.

SDK pour JavaScript (v3)

Orchestrez les étapes (index.js).

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

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
  getManifestFile,
  outputImageSetIds,
```

```
  parseManifestFile,
} from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
      getManifestFile,
      parseManifestFile,
```

```
      outputImageSetIds,
      getImageSetMetadata,
      outputImageFrameIds,
      doVerify,
      decodeAndVerifyImages,
      saveState,
    ],
    context,
  ),
  destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
  ),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Health Imaging Workflow",
    description:
      "Work with DICOM images using an AWS Health Imaging data store.",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-
v|--verbose]",
  });
}
```

Déployez les ressources (deploy-steps.js).

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
```

```
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {{}} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
```

```
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {{}} */ state) => {
    const stackName = state.getStackName;
    const datastoreName = state.getDatastoreName;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreName",
          ParameterValue: datastoreName,
        },
        {
          ParameterKey: "userAccountID",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {{}} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName === state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
```

```
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {{}} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
 string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
    `;
  },
  { skipWhen: (/** @type {{}} */ state) => !state.deployStack },
);
```

Copiez les fichiers DICOM (dataset-steps.js).

```
import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
```

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

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
```

```
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = "input/";
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });
```

```
      return s3Client.send(copyCommand);
    });

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

Lancez l'importation dans l'entrepôt de données (import-steps.js).

```
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
```

```
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (/** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (/** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreID,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
```

```
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  },
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);
```

Obtenez un ensemble d'images IDs (image-set-steps.js-).

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

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

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;
```

```
    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (/** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce((ids, next) => {
        return Object.assign({}, ids, {
          [next.imageSetId]: next.imageSetId,
        });
      }, {});
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (/** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}`,
);
```

Obtenez le cadre d'image IDs (image-frame-steps.js).

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node:util";
```

```
import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
```

```
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (/** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

      outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
```

```
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
        "\n",
      )}\n\n`;
    }

    return output;
  },
);
```

Vérifiez les cadres d'image (verify-steps.js). La bibliothèque AWS HealthImaging Pixel Data Verification a été utilisée pour la vérification.

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

Détruisez les ressources (clean-up-steps.js).

```
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
```

```
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
 PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
```

```
 * @typedef {{
 *  SchemaVersion: string,
 *  DatastoreID: string,
 *  ImageSetID: string,
 *  Patient: Patient,
 *  Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
```

```
          }
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - DeleteImageSet
  - Obtenir un DICOMImport emploi
  - GetImageFrame
  - GetImageSetMetadata
  - SearchImageSets
  - Commencer le DICOMImport Job

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Balisage d'un magasin de données

L'exemple de code suivant montre comment étiqueter un magasin de HealthImaging données.

SDK pour JavaScript (v3)

Pour baliser un magasin de données.

```
  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);
  }
```

Fonction utilitaire permettant de baliser une ressource.

```
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;
};
```

Pour répertorier les balises d'un magasin de données.

```
  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);
  }
```

Fonction utilitaire permettant de répertorier les balises d'une ressource.

```
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;
};
```

Pour supprimer les balises d'un magasin de données.

```
  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);
  }
```

Fonction utilitaire permettant de supprimer les balises d'une ressource.

```
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;
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Balisage d'un ensemble d'images

L'exemple de code suivant montre comment baliser un ensemble HealthImaging d'images.

SDK pour JavaScript (v3)

Pour baliser un ensemble d'images.

```
  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);
  }
```

Fonction utilitaire permettant de baliser une ressource.

```
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;
};
```

Pour répertorier les balises d'un ensemble d'images.

```
  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);
  }
```

Fonction utilitaire permettant de répertorier les balises d'une ressource.

```
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;
};
```

Pour supprimer les balises d'un ensemble d'images.

```
  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);
  }
```

Fonction utilitaire permettant de supprimer les balises d'une ressource.

```
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;
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exemples d'IAM utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec IAM.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour IAM

L'exemple de code suivant montre comment commencer à utiliser IAM.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
 async generators so that you can use async iterators in a for await..of loop.
```

```
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
  for await (const page of paginator) {
    if (page.Policies) {
      for (const policy of page.Policies) {
        console.log(`${policy.PolicyName}`);
        policyCount++;
      }
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- Pour plus de détails sur l'API, reportez-vous ListPoliciesà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant montre comment créer un utilisateur et endosser un rôle.

> **Warning**
>
> Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des données réelles. Préférez la fédération avec un fournisseur d'identité tel que AWS IAM Identity Center.

- Créer un utilisateur sans autorisation.

- Créer un rôle qui accorde l'autorisation de répertorier les compartiments Amazon S3 pour le compte.
- Ajouter une politique pour permettre à l'utilisateur d'assumer le rôle.
- Assumez le rôle et répertorier les compartiments S3 à l'aide d'informations d'identification temporaires, puis nettoyez les ressources.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créer un utilisateur IAM et un rôle qui accorde l'autorisation de répertorier les compartiments Amazon S3. L'utilisateur n'a que le droit d'assumer le rôle. Après avoir assumé le rôle, utilisez des informations d'identification temporaires pour répertorier les compartiments pour le compte.

```
import {
  CreateUserCommand,
  GetUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "iam_basic_test_username";
```

```
const policyName = "iam_basic_test_policy";
const roleName = "iam_basic_test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
      return User;
    }
    throw caught;
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
```

```
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
 STS).
  // It's not best practice to use access keys. For more information, see https://
aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;

  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
      return await listBuckets(s3Client);
    } catch (err) {
      if (err instanceof Error && err.name === "InvalidAccessKeyId") {
        throw err;
      }
    }
  });
```

```
 // Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
 // is thrown while the user and access keys are still stabilizing.
 const { Role } = await retry(
   {
     intervalInMs: 2000,
     maxRetries: 60,
   },
   () =>
     iamClient.send(
       new CreateRoleCommand({
         AssumeRolePolicyDocument: JSON.stringify({
           Version: "2012-10-17",
           Statement: [
             {
               Effect: "Allow",
               Principal: {
                 // Allow the previously created user to assume this role.
                 AWS: User.Arn,
               },
               Action: "sts:AssumeRole",
             },
           ],
         }),
         RoleName: roleName,
       }),
     ),
 );

 if (!Role) {
   throw new Error("Role not created");
 }

 // Create a policy that allows the user to list S3 buckets.
 const { Policy: listBucketPolicy } = await iamClient.send(
   new CreatePolicyCommand({
     PolicyDocument: JSON.stringify({
       Version: "2012-10-17",
       Statement: [
         {
           Effect: "Allow",
           Action: ["s3:ListAllMyBuckets"],
           Resource: "*",
```

```
        },
      ],
    }),
    PolicyName: policyName,
  }),
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
```

```
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 120 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

await iamClient.send(
  new DeletePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
  }),
);

await iamClient.send(
  new DeleteRoleCommand({
    RoleName: Role.RoleName,
  }),
);

await iamClient.send(
  new DeleteAccessKeyCommand({
    UserName: userName,
    AccessKeyId,
  }),
);
```

```
  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Actions

`AttachRolePolicy`

L'exemple de code suivant montre comment utiliser`AttachRolePolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Attachez la politique.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous AttachRolePolicyà la section Référence des AWS SDK pour JavaScript API.

`CreateAccessKey`

L'exemple de code suivant montre comment utiliser`CreateAccessKey`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez la clé d'accès.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateAccessKeyà la section Référence des AWS SDK pour JavaScript API.

`CreateAccountAlias`

L'exemple de code suivant montre comment utiliser`CreateAccountAlias`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez l'alias de compte.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateAccountAliasà la section Référence des AWS SDK pour JavaScript API.

`CreateGroup`

L'exemple de code suivant montre comment utiliser`CreateGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

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

- Pour plus de détails sur l'API, reportez-vous CreateGroupà la section Référence des AWS SDK pour JavaScript API.

`CreateInstanceProfile`

L'exemple de code suivant montre comment utiliser`CreateInstanceProfile`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

- Pour plus de détails sur l'API, reportez-vous CreateInstanceProfileà la section Référence des AWS SDK pour JavaScript API.

`CreatePolicy`

L'exemple de code suivant montre comment utiliser`CreatePolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez la politique.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreatePolicyà la section Référence des AWS SDK pour JavaScript API.

`CreateRole`

L'exemple de code suivant montre comment utiliser`CreateRole`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le rôle.

```
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);
};
```

- Pour plus de détails sur l'API, reportez-vous CreateRoleà la section Référence des AWS SDK pour JavaScript API.

`CreateSAMLProvider`

L'exemple de code suivant montre comment utiliser`CreateSAMLProvider`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import * as path from "node:path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";

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

```
/**
 * This sample document was generated using Auth0.
 * For more information on generating this document,
 see https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_providers_create_saml.html#samlstep1.
 */
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });

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

- Pour plus de détails sur l'API, consultez Create SAMLProvider in AWS SDK pour JavaScript API Reference.

`CreateServiceLinkedRole`

L'exemple de code suivant montre comment utiliser`CreateServiceLinkedRole`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un rôle lié à un service.

```
import {
  CreateServiceLinkedRoleCommand,
  GetRoleCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-
that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/
latest/gr/aws-service-information.html.
    AWSServiceName: serviceName,
  });
  try {
    const response = await client.send(command);
    console.log(response);
    return response;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInputException" &&
      caught.message.includes(
        "Service role name AWSServiceRoleForElasticBeanstalk has been taken in this
 account",
```

```
      )
    ) {
      console.warn(caught.message);
      return client.send(
        new GetRoleCommand({ RoleName: "AWSServiceRoleForElasticBeanstalk" }),
      );
    }
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateServiceLinkedRoleà la section Référence des AWS SDK pour JavaScript API.

`CreateUser`

L'exemple de code suivant montre comment utiliser`CreateUser`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez l'utilisateur.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateUserà la section Référence des AWS SDK pour JavaScript API.

`DeleteAccessKey`

L'exemple de code suivant montre comment utiliser`DeleteAccessKey`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez la clé d'accès.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, reportez-vous DeleteAccessKeyà la section Référence des AWS SDK pour JavaScript API.

`DeleteAccountAlias`

L'exemple de code suivant montre comment utiliser`DeleteAccountAlias`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez l'alias de compte.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteAccountAliasà la section Référence des AWS SDK pour JavaScript API.

`DeleteGroup`

L'exemple de code suivant montre comment utiliser`DeleteGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
    GroupName: groupName,
  });

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

- Pour plus de détails sur l'API, reportez-vous DeleteGroupà la section Référence des AWS SDK pour JavaScript API.

`DeleteInstanceProfile`

L'exemple de code suivant montre comment utiliser`DeleteInstanceProfile`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
```

- Pour plus de détails sur l'API, reportez-vous DeleteInstanceProfileà la section Référence des AWS SDK pour JavaScript API.

`DeletePolicy`

L'exemple de code suivant montre comment utiliser`DeletePolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez la politique.

```
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);
};
```

- Pour plus de détails sur l'API, reportez-vous DeletePolicyà la section Référence des AWS SDK pour JavaScript API.

`DeleteRole`

L'exemple de code suivant montre comment utiliser`DeleteRole`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez le rôle.

```
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);
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteRoleà la section Référence des AWS SDK pour JavaScript API.

`DeleteRolePolicy`

L'exemple de code suivant montre comment utiliser`DeleteRolePolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
  });
  return client.send(command);
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteRolePolicyà la section Référence des AWS SDK pour JavaScript API.

`DeleteSAMLProvider`

L'exemple de code suivant montre comment utiliser`DeleteSAMLProvider`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
  const command = new DeleteSAMLProviderCommand({
    SAMLProviderArn: providerArn,
  });

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

- Pour plus de détails sur l'API, voir Supprimer SAMLProvider dans le manuel de référence des AWS SDK pour JavaScript API.

`DeleteServerCertificate`

L'exemple de code suivant montre comment utiliser`DeleteServerCertificate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez un certificat de serveur.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteServerCertificateà la section Référence des AWS SDK pour JavaScript API.

`DeleteServiceLinkedRole`

L'exemple de code suivant montre comment utiliser`DeleteServiceLinkedRole`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteServiceLinkedRoleà la section Référence des AWS SDK pour JavaScript API.

`DeleteUser`

L'exemple de code suivant montre comment utiliser`DeleteUser`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez l'utilisateur.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteUserà la section Référence des AWS SDK pour JavaScript API.

`DetachRolePolicy`

L'exemple de code suivant montre comment utiliser`DetachRolePolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Détachez la politique.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DetachRolePolicyà la section Référence des AWS SDK pour JavaScript API.

`GetAccessKeyLastUsed`

L'exemple de code suivant montre comment utiliser`GetAccessKeyLastUsed`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez la clé d'accès.

```
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;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetAccessKeyLastUsedà la section Référence des AWS SDK pour JavaScript API.

`GetAccountPasswordPolicy`

L'exemple de code suivant montre comment utiliser`GetAccountPasswordPolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez la politique de mot de passe du compte.

```
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;
};
```

- Pour plus de détails sur l'API, reportez-vous GetAccountPasswordPolicyà la section Référence des AWS SDK pour JavaScript API.

`GetPolicy`

L'exemple de code suivant montre comment utiliser`GetPolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez la politique.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetPolicyà la section Référence des AWS SDK pour JavaScript API.

`GetRole`

L'exemple de code suivant montre comment utiliser`GetRole`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez le rôle.

```
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);
};
```

- Pour plus de détails sur l'API, reportez-vous GetRoleà la section Référence des AWS SDK pour JavaScript API.

`GetServerCertificate`

L'exemple de code suivant montre comment utiliser`GetServerCertificate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez un certificat de serveur.

```
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;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetServerCertificateà la section Référence des AWS SDK pour JavaScript API.

`GetServiceLinkedRoleDeletionStatus`

L'exemple de code suivant montre comment utiliser`GetServiceLinkedRoleDeletionStatus`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
```

```
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

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

- Pour plus de détails sur l'API, reportez-vous GetServiceLinkedRoleDeletionStatusà la section Référence des AWS SDK pour JavaScript API.

`ListAccessKeys`

L'exemple de code suivant montre comment utiliser`ListAccessKeys`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les clés d'accès.

```
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;
    }
  }
}
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListAccessKeysà la section Référence des AWS SDK pour JavaScript API.

`ListAccountAliases`

L'exemple de code suivant montre comment utiliser`ListAccountAliases`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les alias de compte.

```
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;
    }
  }
}
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListAccountAliasesà la section Référence des AWS SDK pour JavaScript API.

`ListAttachedRolePolicies`

L'exemple de code suivant montre comment utiliser`ListAttachedRolePolicies`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les politiques qui sont attachées à un rôle.

```
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;
    }
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListAttachedRolePoliciesà la section Référence des AWS SDK pour JavaScript API.

`ListGroups`

L'exemple de code suivant montre comment utiliser`ListGroups`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les groupes.

```
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;
    }
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListGroupsà la section Référence des AWS SDK pour JavaScript API.

`ListPolicies`

L'exemple de code suivant montre comment utiliser`ListPolicies`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les politiques.

```
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;
    }
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListPoliciesà la section Référence des AWS SDK pour JavaScript API.

`ListRolePolicies`

L'exemple de code suivant montre comment utiliser`ListRolePolicies`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les politiques.

```
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;
    }
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListRolePoliciesà la section Référence des AWS SDK pour JavaScript API.

`ListRoles`

L'exemple de code suivant montre comment utiliser`ListRoles`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les rôles.

```
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;
    }
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListRolesà la section Référence des AWS SDK pour JavaScript API.

`ListSAMLProviders`

L'exemple de code suivant montre comment utiliser`ListSAMLProviders`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les fournisseurs SAML.

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listSamlProviders = async () => {
  const command = new ListSAMLProvidersCommand({});

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

- Pour plus de détails sur l'API, consultez la section Liste SAMLProviders dans la référence des AWS SDK pour JavaScript API.

`ListServerCertificates`

L'exemple de code suivant montre comment utiliser`ListServerCertificates`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les certificats.

```
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;
    }
  }
}
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListServerCertificatesà la section Référence des AWS SDK pour JavaScript API.

`ListUsers`

L'exemple de code suivant montre comment utiliser`ListUsers`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les utilisateurs.

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listUsers = async () => {
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);

  for (const { UserName, CreateDate } of response.Users) {
    console.log(`${UserName} created on: ${CreateDate}`);
  }
  return response;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListUsersà la section Référence des AWS SDK pour JavaScript API.

`PutRolePolicy`

L'exemple de code suivant montre comment utiliser`PutRolePolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::amzn-s3-demo-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
```

```
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

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

- Pour plus de détails sur l'API, reportez-vous PutRolePolicyà la section Référence des AWS SDK pour JavaScript API.

`UpdateAccessKey`

L'exemple de code suivant montre comment utiliser`UpdateAccessKey`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Mettez à jour la clé d'accès.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateAccessKeyà la section Référence des AWS SDK pour JavaScript API.

`UpdateServerCertificate`

L'exemple de code suivant montre comment utiliser`UpdateServerCertificate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Mettez à jour un certificat de serveur.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateServerCertificateà la section Référence des AWS SDK pour JavaScript API.

`UpdateUser`

L'exemple de code suivant montre comment utiliser`UpdateUser`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Mettez à jour l'utilisateur.

```
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);
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateUserà la section Référence des AWS SDK pour JavaScript API.

`UploadServerCertificate`

L'exemple de code suivant montre comment utiliser`UploadServerCertificate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "node:path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
 the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
```

```
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }

    throw err;
  }
};

/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

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

- Pour plus de détails sur l'API, reportez-vous UploadServerCertificateà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer et gérer un service résilient

L'exemple de code suivant montre comment créer un service Web à charge équilibrée qui renvoie des recommandations de livres, de films et de chansons. L'exemple montre comment le service répond aux défaillances et comment le restructurer pour accroître la résilience en cas de défaillance.

- Utilisez un groupe Amazon EC2 Auto Scaling pour créer des instances Amazon Elastic Compute Cloud (Amazon EC2) sur la base d'un modèle de lancement et pour maintenir le nombre d'instances dans une plage spécifiée.
- Gérez et distribuez les requêtes HTTP avec Elastic Load Balancing.

- Surveillez l'état des instances d'un groupe Auto Scaling et transférez les demandes uniquement aux instances saines.
- Exécutez un serveur Web Python sur chaque EC2 instance pour gérer les requêtes HTTP. Le serveur Web répond par des recommandations et des surveillances de l'état.
- Simulez un service de recommandation avec une table Amazon DynamoDB.
- Contrôlez la réponse du serveur Web aux demandes et aux contrôles de santé en mettant à jour AWS Systems Manager les paramètres.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

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

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

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

/**
```

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

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

// Call function if run directly
import { fileURLToPath } from "node:url";

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

Créez des étapes pour déployer toutes les ressources.

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
```

```
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
```

```
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
```

```
          KeyType: "RANGE",
        },
      ],
    }),
  );
  await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
  "createdTable",
  MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "populatingTable",
  MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
```

```
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
```

```
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
```

```
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
```

```
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
```

```
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
```

```
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
```

```
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
  state.targetGroupProtocol = targetGroup.Protocol;
  state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
  "createdLoadBalancerTargetGroup",
  MESSAGES.createdLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
    }),
  );
  state.loadBalancerDns = LoadBalancers[0].DNSName;
  state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
  await waitUntilLoadBalancerAvailable(
    { client },
    { Names: [NAMES.loadBalancerName] },
  );
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
  MESSAGES.createdLoadBalancer
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioOutput(
  "creatingListener",
  MESSAGES.creatingLoadBalancerListener
    .replace("${LB_NAME}", NAMES.loadBalancerName)
```

```
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
),
new ScenarioAction("createListener", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { Listeners } = await client.send(
    new CreateListenerCommand({
      LoadBalancerArn: state.loadBalancerArn,
      Protocol: state.targetGroupProtocol,
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  );
  const listener = Listeners[0];
  state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
  MESSAGES.createdLoadBalancerListener.replace(
    "${LB_LISTENER_ARN}",
    state.loadBalancerListenerArn,
  ),
),
new ScenarioOutput(
  "attachingLoadBalancerTargetGroup",
  MESSAGES.attachingLoadBalancerTargetGroup
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
    .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
),
new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
}),
new ScenarioOutput(
  "attachedLoadBalancerTargetGroup",
  MESSAGES.attachedLoadBalancerTargetGroup,
),
new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
new ScenarioAction(
```

```
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
```

```
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      }
      return MESSAGES.noIpRules;
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      return MESSAGES.noIpRules;
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
```

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

Créez des étapes pour exécuter la démo.

```
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;
}
```

Créez des étapes pour déployer toutes les ressources.

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

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
```

```
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
```

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

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
```

```
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
```

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

```
    NAMES.instanceRoleName,
  );
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
  return MESSAGES.deletedInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  );
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
  try {
    await terminateGroupInstances(NAMES.autoScalingGroupName);
    await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
      await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
    });
  } catch (e) {
    state.deleteAutoScalingGroupError = e;
  }
}),
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
  if (state.deleteAutoScalingGroupError) {
    console.error(state.deleteAutoScalingGroupError);
    return MESSAGES.deleteAutoScalingGroupError.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
```

```
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
```

```
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
```

```
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
```

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

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

```
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
```

```
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
  }),
];

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

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

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

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

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

AWS IoT SiteWise exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec AWS IoT SiteWise.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour AWS IoT SiteWise

L'exemple de code suivant montre comment démarrer avec AWS IoT SiteWise.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  paginateListAssetModels,
  IoTSiteWiseClient,
```

```
} from "@aws-sdk/client-iotsitewise";

// Call ListDocuments and display the result.
export const main = async () => {
  const client = new IoTSiteWiseClient();
  const listAssetModelsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListAssetModels({ client }, { maxResults: 5 });
    for await (const page of paginator) {
      listAssetModelsPaginated.push(...page.assetModelSummaries);
    }
  } catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
  }
  for (const { name, creationDate } of listAssetModelsPaginated) {
    console.log(`${name} - ${creationDate}`);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Pour plus de détails sur l'API, reportez-vous ListAssetModelsà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez un modèle AWS IoT SiteWise d'actifs.
- Créez un AWS IoT SiteWise actif.

- Extrayez les valeurs d'ID de propriété.
- Envoyez des données à un AWS IoT SiteWise actif.
- Récupérez la valeur de la propriété AWS IoT SiteWise Asset.
- Créez un AWS IoT SiteWise portail.
- Créez une AWS IoT SiteWise passerelle.
- Décrivez le AWS IoT SiteWise Gateway.
- Supprimez les AWS IoT SiteWise actifs.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
  //} from "@aws-doc-sdk-examples/lib/scenario/index.js";
} from "../../libs/scenario/index.js";
import {
  IoTSiteWiseClient,
  CreateAssetModelCommand,
  CreateAssetCommand,
  ListAssetModelPropertiesCommand,
  BatchPutAssetPropertyValueCommand,
  GetAssetPropertyValueCommand,
  CreatePortalCommand,
  DescribePortalCommand,
  CreateGatewayCommand,
  DescribeGatewayCommand,
  DeletePortalCommand,
  DeleteGatewayCommand,
  DeleteAssetCommand,
  DeleteAssetModelCommand,
  DescribeAssetModelCommand,
```

```
} from "@aws-sdk/client-iotsitewise";
import {
  CloudFormationClient,
  CreateStackCommand,
  DeleteStackCommand,
  DescribeStacksCommand,
  waitUntilStackExists,
  waitUntilStackCreateComplete,
  waitUntilStackDeleteComplete,
} from "@aws-sdk/client-cloudformation";
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { parseArgs } from "node:util";
import { readFileSync } from "node:fs";
import { fileURLToPath } from "node:url";
import { dirname } from "node:path";

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);
const stackName = "SiteWiseBasicsStack";

/**
 * @typedef {{
 *   iotSiteWiseClient: import('@aws-sdk/client-iotsitewise').IotSiteWiseClient,
 *   cloudFormationClient: import('@aws-sdk/client-
cloudformation').CloudFormationClient,
 *   stackName,
 *   stack,
 *   askToDeleteResources: true,
 *   asset: {assetName: "MyAsset1"},
 *   assetModel: {assetModelName: "MyAssetModel1"},
 *   portal: {portalName: "MyPortal1"},
 *   gateway: {gatewayName: "MyGateway1"},
 *   propertyIds: [],
 *   contactEmail: "user@mydomain.com",
 *   thing: "MyThing1",
 *   sampleData: { temperature: 23.5, humidity: 65.0}
 * }} State
 */

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
```

```
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `AWS IoT SiteWise is a fully managed industrial software-as-a-service (SaaS)
 that makes it easy to collect, store, organize, and monitor data from industrial
 equipment and processes. It is designed to help industrial and manufacturing
 organizations collect data from their equipment and processes, and use that data to
 make informed decisions about their operations.
One of the key features of AWS IoT SiteWise is its ability to connect to a wide
 range of industrial equipment and systems, including programmable logic controllers
 (PLCs), sensors, and other industrial devices. It can collect data from these
 devices and organize it into a unified data model, making it easier to analyze and
 gain insights from the data. AWS IoT SiteWise also provides tools for visualizing
 the data, setting up alarms and alerts, and generating reports.
Another key feature of AWS IoT SiteWise is its ability to scale to handle large
 volumes of data. It can collect and store data from thousands of devices and
 process millions of data points per second, making it suitable for large-scale
 industrial operations. Additionally, AWS IoT SiteWise is designed to be secure
 and compliant, with features like role-based access controls, data encryption,
 and integration with other AWS services for additional security and compliance
 features.

Let's get started...`,
  { header: true },
);

const displayBuildCloudFormationStack = new ScenarioOutput(
  "displayBuildCloudFormationStack",
  "This scenario uses AWS CloudFormation to create an IAM role that is required for
 this scenario. The stack will now be deployed.",
);

const sdkBuildCloudFormationStack = new ScenarioAction(
  "sdkBuildCloudFormationStack",
  async (/** @type {State} */ state) => {
    try {
      const data = readFileSync(
        `${__dirname}/../../../../resources/cfn/iotsitewise_basics/SitewiseRoles-
template.yml`,
        "utf8",
      );
      await state.cloudFormationClient.send(
```

```
        new CreateStackCommand({
          StackName: stackName,
          TemplateBody: data,
          Capabilities: ["CAPABILITY_IAM"],
        }),
      );
      await waitUntilStackExists(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      await waitUntilStackCreateComplete(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      const stack = await state.cloudFormationClient.send(
        new DescribeStacksCommand({
          StackName: stackName,
        }),
      );
      state.stack = stack.Stacks[0].Outputs[0];
      console.log(`The ARN of the IAM role is ${state.stack.OutputValue}`);
    } catch (caught) {
      console.error(caught.message);
      throw caught;
    }
  },
);

const displayCreateAWSSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSSiteWiseAssetModel",
  `1. Create an AWS SiteWise Asset Model
An AWS IoT SiteWise Asset Model is a way to represent the physical assets, such
 as equipment, processes, and systems, that exist in an industrial environment.
 This model provides a structured and hierarchical representation of these assets,
 allowing users to define the relationships and properties of each asset.

This scenario creates two asset model properties: temperature and humidity.`,
);

const sdkCreateAWSSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSSiteWiseAssetModel",
  async (/** @type {State} */ state) => {
    let assetModelResponse;
    try {
```

```
      assetModelResponse = await state.iotSiteWiseClient.send(
        new CreateAssetModelCommand({
          assetModelName: state.assetModel.assetModelName,
          assetModelProperties: [
            {
              name: "Temperature",
              dataType: "DOUBLE",
              type: {
                measurement: {},
              },
            },
            {
              name: "Humidity",
              dataType: "DOUBLE",
              type: {
                measurement: {},
              },
            },
          ],
        }),
      );
      state.assetModel.assetModelId = assetModelResponse.assetModelId;
      console.log(
        `Asset Model successfully created. Asset Model ID:
 ${state.assetModel.assetModelId}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceAlreadyExistsException") {
        console.log(
          `The Asset Model ${state.assetModel.assetModelName} already exists.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayCreateAWSIoTSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSIoTSiteWiseAssetModel",
  `2. Create an AWS IoT SiteWise Asset
The IoT SiteWise model that we just created defines the structure and metadata for
 your physical assets. Now we create an asset from the asset model.
```

```
Let's wait 30 seconds for the asset to be ready.`,
);

const waitThirtySeconds = new ScenarioAction("waitThirtySeconds", async () => {
  await wait(30); // wait 30 seconds
  console.log("Time's up! Let's check the asset's status.");
});

const sdkCreateAWSIoTSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSIoTSiteWiseAssetModel",
  async (/** @type {State} */ state) => {
    try {
      const assetResponse = await state.iotSiteWiseClient.send(
        new CreateAssetCommand({
          assetModelId: state.assetModel.assetModelId,
          assetName: state.asset.assetName,
        }),
      );
      state.asset.assetId = assetResponse.assetId;
      console.log(`Asset created with ID: ${state.asset.assetId}`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(
          `The Asset ${state.assetModel.assetModelName} was not found.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayRetrievePropertyId = new ScenarioOutput(
  "displayRetrievePropertyId",
  `3. Retrieve the property ID values

To send data to an asset, we need to get the property ID values. In this scenario,
 we access the temperature and humidity property ID values.`,
);

const sdkRetrievePropertyId = new ScenarioAction(
  "sdkRetrievePropertyId",
```

```
  async (state) => {
    try {
      const retrieveResponse = await state.iotSiteWiseClient.send(
        new ListAssetModelPropertiesCommand({
          assetModelId: state.assetModel.assetModelId,
        }),
      );
      for (const retrieveResponseKey in
 retrieveResponse.assetModelPropertySummaries) {
        if (
          retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
            .name === "Humidity"
        ) {
          state.propertyIds.Humidity =
            retrieveResponse.assetModelPropertySummaries[
              retrieveResponseKey
            ].id;
        }
        if (
          retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
            .name === "Temperature"
        ) {
          state.propertyIds.Temperature =
            retrieveResponse.assetModelPropertySummaries[
              retrieveResponseKey
            ].id;
        }
      }
      console.log(`The Humidity propertyId is ${state.propertyIds.Humidity}`);
      console.log(
        `The Temperature propertyId is ${state.propertyIds.Temperature}`,
      );
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem retrieving the properties: ${caught.message}`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);
```

```
const displaySendDataToIoTSiteWiseAsset = new ScenarioOutput(
  "displaySendDataToIoTSiteWiseAsset",
  `4. Send data to an AWS IoT SiteWise Asset

By sending data to an IoT SiteWise Asset, you can aggregate data from multiple
 sources, normalize the data into a standard format, and store it in a centralized
 location. This makes it easier to analyze and gain insights from the data.

In this example, we generate sample temperature and humidity data and send it to the
 AWS IoT SiteWise asset.`,
);

const sdkSendDataToIoTSiteWiseAsset = new ScenarioAction(
  "sdkSendDataToIoTSiteWiseAsset",
  async (state) => {
    try {
      const sendResponse = await state.iotSiteWiseClient.send(
        new BatchPutAssetPropertyValueCommand({
          entries: [
            {
              entryId: "entry-3",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Humidity,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.humidity,
                  },
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
            {
              entryId: "entry-4",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Temperature,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.temperature,
                  },
```

```
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
          ],
        }),
      );
      console.log("The data was sent successfully.");
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayRetrieveValueOfIoTSiteWiseAsset = new ScenarioOutput(
  "displayRetrieveValueOfIoTSiteWiseAsset",
  `5. Retrieve the value of the IoT SiteWise Asset property

IoT SiteWise is an AWS service that allows you to collect, process, and analyze
 industrial data from connected equipment and sensors. One of the key benefits of
 reading an IoT SiteWise property is the ability to gain valuable insights from your
 industrial data.`,
);

const sdkRetrieveValueOfIoTSiteWiseAsset = new ScenarioAction(
  "sdkRetrieveValueOfIoTSiteWiseAsset",
  async (/** @type {State} */ state) => {
    try {
      const temperatureResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
          propertyId: state.propertyIds.Temperature,
        }),
      );
      const humidityResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
```

```
          propertyId: state.propertyIds.Humidity,
        }),
      );
      console.log(
        `The property value for Temperature is
 ${temperatureResponse.propertyValue.value.doubleValue}`,
      );
      console.log(
        `The property value for Humidity is
 ${humidityResponse.propertyValue.value.doubleValue}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayCreateIoTSiteWisePortal = new ScenarioOutput(
  "displayCreateIoTSiteWisePortal",
  `6. Create an IoT SiteWise Portal

An IoT SiteWise Portal allows you to aggregate data from multiple industrial
 sources, such as sensors, equipment, and control systems, into a centralized
 platform.`,
);

const sdkCreateIoTSiteWisePortal = new ScenarioAction(
  "sdkCreateIoTSiteWisePortal",
  async (/** @type {State} */ state) => {
    try {
      const createPortalResponse = await state.iotSiteWiseClient.send(
        new CreatePortalCommand({
          portalName: state.portal.portalName,
          portalContactEmail: state.contactEmail,
          roleArn: state.stack.OutputValue,
        }),
      );
      state.portal = { ...state.portal, ...createPortalResponse };
      await wait(5); // Allow the portal to properly propagate.
```

```
      console.log(
        `Portal created successfully. Portal ID ${createPortalResponse.portalId}`,
      );
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem creating the Portal: ${caught.message}.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayDescribePortal = new ScenarioOutput(
  "displayDescribePortal",
  `7. Describe the Portal

In this step, we get a description of the portal and display the portal URL.`,
);

const sdkDescribePortal = new ScenarioAction(
  "sdkDescribePortal",
  async (/** @type {State} */ state) => {
    try {
      const describePortalResponse = await state.iotSiteWiseClient.send(
        new DescribePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(`Portal URL: ${describePortalResponse.portalStartUrl}`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);
```

```
const displayCreateIoTSiteWiseGateway = new ScenarioOutput(
  "displayCreateIoTSiteWiseGateway",
  `8. Create an IoT SiteWise Gateway

IoT SiteWise Gateway serves as the bridge between industrial equipment, sensors, and
 the cloud-based IoT SiteWise service. It is responsible for securely collecting,
 processing, and transmitting data from various industrial assets to the IoT
 SiteWise platform, enabling real-time monitoring, analysis, and optimization of
 industrial operations.`,
);

const sdkCreateIoTSiteWiseGateway = new ScenarioAction(
  "sdkCreateIoTSiteWiseGateway",
  async (/** @type {State} */ state) => {
    try {
      const createGatewayResponse = await state.iotSiteWiseClient.send(
        new CreateGatewayCommand({
          gatewayName: state.gateway.gatewayName,
          gatewayPlatform: {
            greengrassV2: {
              coreDeviceThingName: state.thing,
            },
          },
        }),
      );
      console.log(
        `Gateway creation completed successfully. ID is
 ${createGatewayResponse.gatewayId}`,
      );
      state.gateway.gatewayId = createGatewayResponse.gatewayId;
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem creating the gateway: ${caught.message}.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayDescribeIoTSiteWiseGateway = new ScenarioOutput(
```

```
  "displayDescribeIoTSiteWiseGateway",
  "9. Describe the IoT SiteWise Gateway",
);

const sdkDescribeIoTSiteWiseGateway = new ScenarioAction(
  "sdkDescribeIoTSiteWiseGateway",
  async (/** @type {State} */ state) => {
    try {
      const describeGatewayResponse = await state.iotSiteWiseClient.send(
        new DescribeGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log("Gateway creation completed successfully.");
      console.log(`Gateway Name: ${describeGatewayResponse.gatewayName}`);
      console.log(`Gateway ARN: ${describeGatewayResponse.gatewayArn}`);
      console.log(
        `Gateway Platform: ${Object.keys(describeGatewayResponse.gatewayPlatform)}`,
      );
      console.log(
        `Gateway Creation Date: ${describeGatewayResponse.creationDate}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const askToDeleteResources = new ScenarioInput(
  "askToDeleteResources",
  `10. Delete the AWS IoT SiteWise Assets

Before you can delete the Asset Model, you must delete the assets.`,
  { type: "confirm" },
);

const displayConfirmDeleteResources = new ScenarioAction(
  "displayConfirmDeleteResources",
  async (/** @type {State} */ state) => {
```

```
    if (state.askToDeleteResources) {
      return "You selected to delete the SiteWise assets.";
    }
    return "The resources will not be deleted. Please delete them manually to avoid
 charges.";
  },
);

const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    await wait(10); // Give the portal status time to catch up.
    try {
      await state.iotSiteWiseClient.send(
        new DeletePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(
        `Portal ${state.portal.portalName} was deleted successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
      } else {
        console.log(`When trying to delete the portal: ${caught.message}`);
      }
    }

    try {
      await state.iotSiteWiseClient.send(
        new DeleteGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log(
        `Gateway ${state.gateway.gatewayName} was deleted successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
      } else {
        console.log(`When trying to delete the gateway: ${caught.message}`);
      }
```

```
}

try {
  await state.iotSiteWiseClient.send(
    new DeleteAssetCommand({
      assetId: state.asset.assetId,
    }),
  );
  await wait(5); // Allow the delete to finish.
  console.log(`Asset ${state.asset.assetName} was deleted successfully.`);
} catch (caught) {
  if (caught.name === "ResourceNotFoundException") {
    console.log(`The Asset ${state.asset.assetName} was not found.`);
  } else {
    console.log(`When deleting the asset: ${caught.message}`);
  }
}

await wait(30); // Allow asset deletion to finish.
try {
  await state.iotSiteWiseClient.send(
    new DeleteAssetModelCommand({
      assetModelId: state.assetModel.assetModelId,
    }),
  );
  console.log(
    `Asset Model ${state.assetModel.assetModelName} was deleted successfully.`,
  );
} catch (caught) {
  if (caught.name === "ResourceNotFoundException") {
    console.log(
      `The Asset Model ${state.assetModel.assetModelName} was not found.`,
    );
  } else {
    console.log(`When deleting the asset model: ${caught.message}`);
  }
}

try {
  await state.cloudFormationClient.send(
    new DeleteStackCommand({
      StackName: stackName,
    }),
  );
```

```
      await waitUntilStackDeleteComplete(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      console.log("The stack was deleted successfully.");
    } catch (caught) {
      console.log(
        `${caught.message}. The stack was NOT deleted. Please clean up the resources
 manually.`,
      );
    }
  },
  { skipWhen: (/** @type {{}} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the IoT Sitewise Basics scenario for the AWS Javascript SDK v3.
 Thank you!",
);

const myScenario = new Scenario(
  "IoTSiteWise Basics",
  [
    greet,
    pressEnter,
    displayBuildCloudFormationStack,
    sdkBuildCloudFormationStack,
    pressEnter,
    displayCreateAWSSiteWiseAssetModel,
    sdkCreateAWSSiteWiseAssetModel,
    displayCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    waitThirtySeconds,
    sdkCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    displayRetrievePropertyId,
    sdkRetrievePropertyId,
    pressEnter,
    displaySendDataToIoTSiteWiseAsset,
    sdkSendDataToIoTSiteWiseAsset,
    pressEnter,
    displayRetrieveValueOfIoTSiteWiseAsset,
    sdkRetrieveValueOfIoTSiteWiseAsset,
```

```
    pressEnter,
    displayCreateIoTSiteWisePortal,
    sdkCreateIoTSiteWisePortal,
    pressEnter,
    displayDescribePortal,
    sdkDescribePortal,
    pressEnter,
    displayCreateIoTSiteWiseGateway,
    sdkCreateIoTSiteWiseGateway,
    pressEnter,
    displayDescribeIoTSiteWiseGateway,
    sdkDescribeIoTSiteWiseGateway,
    pressEnter,
    askToDeleteResources,
    displayConfirmDeleteResources,
    sdkDeleteResources,
    goodbye,
  ],
  {
    iotSiteWiseClient: new IoTSiteWiseClient({}),
    cloudFormationClient: new CloudFormationClient({}),
    asset: { assetName: "MyAsset1" },
    assetModel: { assetModelName: "MyAssetModel1" },
    portal: { portalName: "MyPortal1" },
    gateway: { gatewayName: "MyGateway1" },
    propertyIds: [],
    contactEmail: "user@mydomain.com",
    thing: "MyThing1",
    sampleData: { temperature: 23.5, humidity: 65.0 },
  },
);

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

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

```
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

Actions

`BatchPutAssetPropertyValue`

L'exemple de code suivant montre comment utiliser`BatchPutAssetPropertyValue`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  BatchPutAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Batch put asset property values.
 * @param {{ entries : array }}
 */
export const main = async ({ entries }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new BatchPutAssetPropertyValueCommand({
        entries: entries,
      }),
    );
    console.log("Asset properties batch put successfully.");
```

```
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(`${caught.message}. A resource could not be found.`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous BatchPutAssetPropertyValueà la section Référence des AWS SDK pour JavaScript API.

`CreateAsset`

L'exemple de code suivant montre comment utiliser`CreateAsset`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  CreateAssetCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an Asset.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetName, assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateAssetCommand({
```

```
        assetName: assetName, // The name to give the Asset.
        assetModelId: assetModelId, // The ID of the asset model from which to
 create the asset.
      }),
    );
    console.log("Asset created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset model could not be found. Please check the
 asset model id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateAssetà la section Référence des AWS SDK pour JavaScript API.

`CreateAssetModel`

L'exemple de code suivant montre comment utiliser`CreateAssetModel`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  CreateAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";
```

```
/**
 * Create an Asset Model.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetModelName, assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateAssetModelCommand({
        assetModelName: assetModelName, // The name to give the Asset Model.
      }),
    );
    console.log("Asset model created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the asset model.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateAssetModelà la section Référence des AWS SDK pour JavaScript API.

`CreateGateway`

L'exemple de code suivant montre comment utiliser`CreateGateway`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  CreateGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create a Gateway.
 * @param {{  }}
 */
export const main = async ({ gatewayName }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateGatewayCommand({
        gatewayName: gatewayName, // The name to give the created Gateway.
      }),
    );
    console.log("Gateway created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the Gateway.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateGatewayà la section Référence des AWS SDK pour JavaScript API.

`DeleteAsset`

L'exemple de code suivant montre comment utiliser`DeleteAsset`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  DeleteAssetCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset.
 * @param {{ assetId : string }}
 */
export const main = async ({ assetId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetCommand({
        assetId: assetId, // The model id to delete.
      }),
    );
    console.log("Asset deleted successfully.");
    return { assetDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteAssetà la section Référence des AWS SDK pour JavaScript API.

`DeleteAssetModel`

L'exemple de code suivant montre comment utiliser`DeleteAssetModel`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  DeleteAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetModelCommand({
        assetModelId: assetModelId, // The model id to delete.
      }),
    );
    console.log("Asset model deleted successfully.");
    return { assetModelDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset model.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteAssetModelà la section Référence des AWS SDK pour JavaScript API.

`DeleteGateway`

L'exemple de code suivant montre comment utiliser`DeleteGateway`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  DeleteGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway deleted successfully.");
    return { gatewayDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
```

```
        `${caught.message}. The Gateway could not be found. Please check the Gateway
 Id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteGatewayà la section Référence des AWS SDK pour JavaScript API.

`DescribeAssetModel`

L'exemple de code suivant montre comment utiliser`DescribeAssetModel`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  DescribeAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const { assetModelDescription } = await client.send(
      new DescribeAssetModelCommand({
        assetModelId: assetModelId, // The ID of the Gateway to describe.
```

```
      }),
    );
    console.log("Asset model information retrieved successfully.");
    return { assetModelDescription: assetModelDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset model could not be found. Please check the
 asset model id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeAssetModelà la section Référence des AWS SDK pour JavaScript API.

`DescribeGateway`

L'exemple de code suivant montre comment utiliser`DescribeGateway`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  DescribeGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
```

```
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const { gatewayDescription } = await client.send(
      new DescribeGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway information retrieved successfully.");
    return { gatewayDescription: gatewayDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Gateway could not be found. Please check the Gateway
 Id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeGatewayà la section Référence des AWS SDK pour JavaScript API.

`GetAssetPropertyValue`

L'exemple de code suivant montre comment utiliser`GetAssetPropertyValue`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  GetAssetPropertyValueCommand,
```

```
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset property value.
 * @param {{ entryId : string }}
 */
export const main = async ({ entryId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new GetAssetPropertyValueCommand({
        entryId: entryId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Asset property information retrieved successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset property entry could not be found. Please
 check the entry id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous GetAssetPropertyValueà la section Référence des AWS SDK pour JavaScript API.

`ListAssetModels`

L'exemple de code suivant montre comment utiliser`ListAssetModels`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  ListAssetModelsCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * List asset models.
 * @param {{ assetModelTypes : array }}
 */
export const main = async ({ assetModelTypes = [] }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new ListAssetModelsCommand({
        assetModelTypes: assetModelTypes, // The model types to list
      }),
    );
    console.log("Asset model types retrieved successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem listing the asset model types.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ListAssetModelsà la section Référence des AWS SDK pour JavaScript API.

Exemples Kinesis utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Kinesis.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Actions

`PutRecords`

L'exemple de code suivant montre comment utiliser`PutRecords`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { PutRecordsCommand, KinesisClient } from "@aws-sdk/client-kinesis";

/**
 * Put multiple records into a Kinesis stream.
 * @param {{ streamArn: string }} config
 */
export const main = async ({ streamArn }) => {
  const client = new KinesisClient({});
  try {
```

```
    await client.send(
      new PutRecordsCommand({
        StreamARN: streamArn,
        Records: [
          {
            Data: new Uint8Array(),
            /**
             * Determines which shard in the stream the data record is assigned to.
             * Partition keys are Unicode strings with a maximum length limit of 256
             * characters for each key. Amazon Kinesis Data Streams uses the
 partition
             * key as input to a hash function that maps the partition key and
             * associated data to a specific shard.
             */
            PartitionKey: "TEST_KEY",
          },
          {
            Data: new Uint8Array(),
            PartitionKey: "TEST_KEY",
          },
        ],
      }),
    );
  } catch (caught) {
    if (caught instanceof Error) {
      //
    } else {
      throw caught;
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    streamArn: {
      type: "string",
      description: "The ARN of the stream.",
    },
  };
```

```
  const { values } = parseArgs({ options });
  main(values);
}
```

- Pour plus de détails sur l'API, reportez-vous PutRecordsà la section Référence des AWS SDK pour JavaScript API.

Exemples sans serveur

Invoquer une fonction Lambda à partir d'un déclencheur Kinesis

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux Kinesis. La fonction récupère la charge utile Kinesis, décode à partir de Base64 et enregistre le contenu de l'enregistrement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de. 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;
}
```

Utilisation d'un événement Kinesis avec Lambda à l'aide de. 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;
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Kinesis

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'un flux Kinesis. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
```

```
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de. 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;
}
```

Exemples Lambda utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec Lambda.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Hello Lambda

L'exemple de code suivant montre comment commencer à utiliser Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
```

```
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Pour plus de détails sur l'API, reportez-vous ListFunctionsà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créer un rôle IAM et une fonction Lambda, puis charger le code du gestionnaire.
- Invoquer la fonction avec un seul paramètre et obtenir des résultats.
- Mettre à jour le code de la fonction et configurer avec une variable d'environnement.
- Invoquer la fonction avec de nouveaux paramètres et obtenir des résultats. Afficher le journal d'exécution renvoyé.
- Répertorier les fonctions pour votre compte, puis nettoyer les ressources.

Pour plus d'informations, consultez Créer une fonction Lambda à l'aide de la console.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un rôle Gestion des identités et des accès AWS (IAM) qui accorde à Lambda l'autorisation d'écrire dans les journaux.

```
    logger.log(`Creating role (${NAME_ROLE_LAMBDA})...`);
    const response = await createRole(NAME_ROLE_LAMBDA);
```

```
import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

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

Créer une fonction Lambda et télécharger le code de gestionnaire.

```
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);
};
```

Invoquer la fonction avec un seul paramètre et obtenir des résultats.

```
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 };
};
```

Mettre à jour le code de fonction et configurer son environnement Lambda avec une variable d'environnement.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

Répertorier les fonctions pour votre compte.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

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

Supprimer le rôle IAM et la fonction Lambda.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript .
  - CreateFunction
  - DeleteFunction

- GetFunction
- Invoke
- ListFunctions
- UpdateFunctionCode
- UpdateFunctionConfiguration

Actions

`CreateFunction`

L'exemple de code suivant montre comment utiliser`CreateFunction`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

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

- Pour plus de détails sur l'API, reportez-vous CreateFunctionà la section Référence des AWS SDK pour JavaScript API.

`DeleteFunction`

L'exemple de code suivant montre comment utiliser`DeleteFunction`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteFunctionà la section Référence des AWS SDK pour JavaScript API.

`GetFunction`

L'exemple de code suivant montre comment utiliser`GetFunction`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Pour plus de détails sur l'API, reportez-vous GetFunctionà la section Référence des AWS SDK pour JavaScript API.

`Invoke`

L'exemple de code suivant montre comment utiliser`Invoke`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

- Pour en savoir plus sur l'API, consultez Invoke dans la Référence de l'API AWS SDK pour JavaScript .

`ListFunctions`

L'exemple de code suivant montre comment utiliser`ListFunctions`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

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

- Pour plus de détails sur l'API, reportez-vous ListFunctionsà la section Référence des AWS SDK pour JavaScript API.

`UpdateFunctionCode`

L'exemple de code suivant montre comment utiliser`UpdateFunctionCode`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
```

```
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

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

- Pour plus de détails sur l'API, reportez-vous UpdateFunctionCodeà la section Référence des AWS SDK pour JavaScript API.

`UpdateFunctionConfiguration`

L'exemple de code suivant montre comment utiliser`UpdateFunctionConfiguration`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

- Pour plus de détails sur l'API, reportez-vous UpdateFunctionConfigurationà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Confirmation automatique des utilisateurs connus avec une fonction Lambda

L'exemple de code suivant illustre comment confirmer automatiquement les utilisateurs Amazon Cognito connus avec une fonction Lambda.

- Configurez un groupe d'utilisateurs pour appeler une fonction Lambda pour le déclencheur `PreSignUp`.
- Inscription d'un utilisateur avec Amazon Cognito.
- La fonction Lambda analyse une table DynamoDB et confirme automatiquement les utilisateurs connus.
- Connectez-vous en tant que nouvel utilisateur, puis nettoyez les ressources.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Configurez une exécution interactive de type « Scénario ». Les exemples JavaScript (v3) partagent un générateur de scénarios pour rationaliser les exemples complexes. Le code source complet est activé GitHub.

```
import { AutoConfirm } from "./scenario-auto-confirm.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      UserEmail: "test_email_1@example.com",
    },
```

```
    {
      UserName: "test_user_2",
      UserEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      UserEmail: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito
 authentication behavior.",
  });
}
```

Ce scénario illustre la confirmation automatique d'un utilisateur connu. Il orchestre les étapes de l'exemple.

```
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
```

```
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
} from "./steps-common.js";
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

const greeting = new ScenarioOutput(
  "greeting",
```

```
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
```

```
  async (/** @type {State} */ state) => {
    const [_, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
 "${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (/** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (/** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (/** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });
```

```
    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    if (user) {
      state.errors.push(
        new Error(
          `The user "${state.selectedUser}" already exists in the user pool
 "${state.UserPoolId}".`,
        ),
      );
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
 numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (/** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({
```

```
        region: state.stackRegion,
        userPoolClientId: state.UserPoolClientId,
        username: state.selectedUser,
        email: state.users.find((u) => u.UserName === state.selectedUser)
          .UserEmail,
        password,
      });

    let [_, err] = await signUp(state.password);

    while (err?.name === "InvalidPasswordException") {
      console.warn("The password you entered was invalid.");
      await createPassword.handle(state);
      [_, err] = await signUp(state.password);
    }

    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (/** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
    if (logStreamErr) {
      state.errors.push(logStreamErr);
```

```
      return;
    }

    console.log(
      `Getting some recent events from log stream "${logStream.logStreamName}"`,
    );
    const [logEvents, logEventsErr] = await getLogEvents({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
      eventCount: 10,
      logStreamName: logStream.logStreamName,
    });
    if (logEventsErr) {
      state.errors.push(logEventsErr);
      return;
    }

    console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });

    if (err?.name === "PasswordResetRequiredException") {
      state.errors.push(new Error("Please reset your password."));
      return;
    }

    if (err) {
```

```
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with: ${state.token.slice(0,
 11)}`,
  { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [_, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);

const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State}*/ state) => {
```

```
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
  );
```

Ces étapes sont partagées avec d'autres scénarios.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
```

```
  { skipWhen: skipWhenErrors },
);
```

Un gestionnaire pour le déclencheur `PreSignUp` avec une fonction Lambda.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "./user-repository";
import { DynamoDBUserRepository } from "./user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user '${event.userName}'`,
    );

    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }

    const eventEmail = this.getEventUserEmail(event);
    console.log(`Looking up email ${eventEmail}.`);
    const storedUserInfo =
      await this.userRepository.getUserInfoByEmail(eventEmail);

    if (!storedUserInfo) {
      console.log(
```

```
        `Email ${eventEmail} not found. Email verification is required.`,
      );
      return event;
    }

    if (storedUserInfo.UserName !== event.userName) {
      console.log(
        `UserEmail ${eventEmail} found, but stored UserName
 '${storedUserInfo.UserName}' does not match supplied UserName '${event.userName}'.
 Verification is required.`,
      );
    } else {
      console.log(
        `UserEmail ${eventEmail} found with matching UserName
 ${storedUserInfo.UserName}. User is confirmed.`,
      );
      event.response.autoConfirmUser = true;
      event.response.autoVerifyEmail = true;
    }
    return event;
  }
}

const createPreSignUpHandler = (): PreSignUpHandler => {
  const tableName = process.env.TABLE_NAME;
  if (!tableName) {
    throw new Error("TABLE_NAME environment variable is not set");
  }

  const userRepository = new DynamoDBUserRepository(tableName);
  return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

Module d'actions de CloudWatch journalisation.

```
import {
```

```
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
 unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[] |
 null, unknown]>}
```

```
 */
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};
```

Module d’actions Amazon Cognito.

```
import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
```

```
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 *   password: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
```

```
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const response = await cognitoClient.send(
      new SignUpCommand({
        ClientId: userPoolClientId,
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 flow.
 * @param {{ region: string, clientId: string, username: string, password: string }}
 config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Retrieve an existing user from a user pool.
```

```
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

Module d’actions DynamoDB.

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

```
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }}
 config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
 null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Création d’une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour JavaScript (v3)

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l’aide d’Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l’origine de cet exemple, consultez l’article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Créez une application pour analyser les commentaires des clients

L’exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d’origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour JavaScript (v3)

Cet exemple d’application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d’un hôtel fictif situé à New York. L’hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l’application via un client Web. Après avoir chargé l’image d’une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub. Les extraits suivants montrent comment le AWS SDK pour JavaScript est utilisé dans les fonctions Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);
```

```
  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
```

```
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Invocation d’une fonction Lambda à partir d’un navigateur

L'exemple de code suivant montre comment invoquer une AWS Lambda fonction depuis un navigateur.

SDK pour JavaScript (v3)

Vous pouvez créer une application basée sur un navigateur qui utilise une AWS Lambda fonction pour mettre à jour une table Amazon DynamoDB avec les sélections des utilisateurs. Cette application utilise la AWS SDK pour JavaScript version 3.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda

Utiliser API Gateway pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par Amazon API Gateway.

SDK pour JavaScript (v3)

Montre comment créer une AWS Lambda fonction à l'aide de l'API JavaScript d'exécution Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une fonction Lambda invoquée par Amazon API Gateway qui analyse une table Amazon DynamoDB à la recherche d'anniversaires professionnels et utilise Amazon Simple Notification Service (Amazon SNS) pour envoyer un message texte à vos employés qui les félicitent à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Utilisent des événements planifiés pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par un événement EventBridge planifié par Amazon.

SDK pour JavaScript (v3)

Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Exemples sans serveur

Connexion à une base de données Amazon RDS dans une fonction Lambda

L'exemple de code suivant montre comment implémenter une fonction Lambda qui se connecte à une base de données RDS. La fonction effectue une simple requête de base de données et renvoie le résultat.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une base de données Amazon RDS dans une fonction Lambda à l'aide de. 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)
  }
};
```

Connexion à une base de données Amazon RDS dans une fonction Lambda à l'aide de. 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}`)
```

```
    };
};
```

Invoquer une fonction Lambda à partir d'un déclencheur Kinesis

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux Kinesis. La fonction récupère la charge utile Kinesis, décode à partir de Base64 et enregistre le contenu de l'enregistrement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de. 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;
```

```
}
```

Utilisation d'un événement Kinesis avec Lambda à l'aide de. 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;
}
```

Invocation d'une fonction Lambda à partir d'un déclencheur DynamoDB

L'exemple de code suivant montre comment mettre en œuvre une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux DynamoDB. La fonction récupère les données utiles DynamoDB et journalise le contenu de l'enregistrement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. 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)}`);
};
```

Consommation d'un événement DynamoDB avec Lambda en utilisant. 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)}`);
};
```

Invocation d'une fonction Lambda à partir d'un déclencheur Amazon DocumentDB

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux de modifications DocumentDB. La fonction récupère les données utiles DocumentDB et journalise le contenu de l'enregistrement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de. 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));
```

```
};
```

Utilisation d'un événement Amazon DocumentDB avec Lambda à l'aide de 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));
};
```

Invocation d’une fonction Lambda à partir d’un déclencheur Amazon MSK

L’exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d’enregistrements à partir d’un cluster Amazon MSK. La fonction récupère les données utiles MSK et journalise le contenu de l’enregistrement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples sans serveur.

Utilisation d'un événement Amazon MSK avec JavaScript Lambda à l'aide de.

```
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)
      })
    }
}
```

Utilisation d'un événement Amazon MSK avec TypeScript Lambda à l'aide de.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
```

```
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    };
  }
}
```

Invoquer une fonction lambda à partir d'un déclencheur Amazon S3

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par le chargement d'un objet vers un compartiment S3. La fonction extrait le nom du compartiment S3 et la clé de l'objet à partir du paramètre d'événement et appelle l'API Amazon S3 pour récupérer et consigner le type de contenu de l'objet.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda en utilisant. 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);
    }
};
```

Consommation d'un événement S3 avec Lambda en utilisant. 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);
  }
};
```

Invocation d'une fonction lambda à partir d'un déclencheur Amazon SNS

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d'une rubrique SNS. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement SNS avec JavaScript Lambda en utilisant.

```
// 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;
  }
}
```

Consommation d'un événement SNS avec TypeScript Lambda en utilisant.

```
// 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;
  }
}
```

Invoquer une fonction Lambda à partir d’un déclencheur Amazon SQS

L’exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d’une file d’attente SQS. La fonction extrait les messages du paramètre d’événement et consigne le contenu de chaque message.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples sans serveur.

Consommation d'un événement SQS avec JavaScript Lambda en utilisant.

```
// 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;
  }
}
```

Consommation d'un événement SQS avec TypeScript Lambda en utilisant.

```
// 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;
  }
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Kinesis

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'un flux Kinesis. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed item
 onwards. */
```

```
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de. 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;
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'un flux DynamoDB. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. 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: [] };
};
```

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. 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 };
};
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'une file d'attente SQS. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot SQS avec JavaScript Lambda à l'aide de.

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

```
}
```

Signaler les défaillances d'éléments de lot SQS avec TypeScript Lambda à l'aide de.

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

Exemples d'Amazon Lex utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Lex.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d’un même service ou combinés à d’autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Scénarios

Scénarios

Création d'un chatbot Amazon Lex

L'exemple de code suivant montre comment créer un chatbot pour interagir avec les visiteurs de votre site Web.

SDK pour JavaScript (v3)

Indique comment utiliser l'API Amazon Lex pour créer un chatbot dans une application Web afin d'interagir avec les visiteurs de votre site Web.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet de création d'un chatbot Amazon Lex dans le guide du AWS SDK pour JavaScript développeur.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Exemples d'Amazon Location utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la AWS SDK pour JavaScript version (v3) avec Amazon Location.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour Amazon Location

L'exemple de code suivant montre comment commencer à utiliser Amazon Location Service.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  LocationClient,
  ListGeofenceCollectionsCommand,
} from "@aws-sdk/client-location";

/**
 * Lists geofences from a specified geofence collection asynchronously.
 */
export const main = async () => {
  const region = "eu-west-1";
  const locationClient = new LocationClient({ region: region });
  const listGeofenceCollParams = {
    MaxResults: 100,
  };
  try {
    const command = new ListGeofenceCollectionsCommand(listGeofenceCollParams);
    const response = await locationClient.send(command);
```

```
    const geofenceEntries = response.Entries;
    if (geofenceEntries.length === 0) {
      console.log("No Geofences were found in the collection.");
    } else {
      for (const geofenceEntry of geofenceEntries) {
        console.log(`Geofence ID: ${geofenceEntry.CollectionName}`);
      }
    }
  } catch (error) {
    console.error(
      `A validation error occurred while creating geofence: ${error} \n Exiting
 program.`,
    );
    return;
  }
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - ListGeofenceCollections
  - ListGeofences

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez une carte Amazon Location.
- Créez une clé d'API Amazon Location.
- Affichez l'URL de la carte.
- Créez une collection de périmètres virtuels.
- Stockez une géométrie de périmètres virtuels.
- créee une ressource de suivi ;
- Mettez à jour la position d'un appareil.
- Extrayez la dernière mise à jour de la position pour un appareil spécifié.

- Créez un calculateur d'itinéraire.
- déterminer la distance entre Seattle et Vancouver ;
- Utilisez le niveau supérieur d'Amazon Location APIs.
- Supprimez les ressources Amazon Location.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/*
Before running this JavaScript code example, set up your development environment,
 including your credentials.
This demo illustrates how to use the AWS SDK for JavaScript (v3) to work with Amazon
 Location Service.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/getting-
started.html
*/

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

import {
  CreateMapCommand,
  CreateGeofenceCollectionCommand,
  PutGeofenceCommand,
  CreateTrackerCommand,
  BatchUpdateDevicePositionCommand,
  GetDevicePositionCommand,
```

```
  CreateRouteCalculatorCommand,
  CalculateRouteCommand,
  LocationClient,
  ConflictException,
  ResourceNotFoundException,
  DeleteGeofenceCollectionCommand,
  DeleteRouteCalculatorCommand,
  DeleteTrackerCommand,
  DeleteMapCommand,
} from "@aws-sdk/client-location";

import {
  GeoPlacesClient,
  ReverseGeocodeCommand,
  SearchNearbyCommand,
  SearchTextCommand,
  GetPlaceCommand,
  ValidationException,
} from "@aws-sdk/client-geo-places";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";

/*The inputs for this example can be edited in the ./input.json.*/
import data from "./inputs.json" with { type: "json" };

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
/* v8 ignore next 3 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
  verbose: "false",
});

const pressEnterConfirm = new ScenarioInput(
  "confirm",
  "Press Enter to continue",
  {
    type: "confirm",
    verbose: "false",
  },
);
```

```
const region = "eu-west-1";

const locationClient = new LocationClient({ region: region });

const greet = new ScenarioOutput(
  "greet",
  "Welcome to the Amazon Location Use demo! \n" +
    "AWS Location Service is a fully managed service offered by Amazon Web Services
 (AWS) that " +
    "provides location-based services for developers. This service simplifies " +
    "the integration of location-based features into applications, making it " +
    "Maps: The service provides access to high-quality maps, satellite imagery, " +
    "and geospatial data from various providers, allowing developers to " +
    "easily embed maps into their applications:\n" +
    "Tracking: The Location Service enables real-time tracking of mobile devices, "
 +
    "assets, or other entities, allowing developers to build applications " +
    "that can monitor the location of people, vehicles, or other objects.\n" +
    "Geocoding: The service provides the ability to convert addresses or " +
    "location names into geographic coordinates (latitude and longitude), " +
    "and vice versa, enabling developers to integrate location-based search " +
    "and routing functionality into their applications. " +
    "Please define values ./inputs.json for each user-defined variable used in this
 app. Otherwise the default is used:\n" +
    "- mapName: The name of the map to be create (default is 'AWSMap').\n" +
    "- keyName: The name of the API key to create (default is ' AWSApiKey')\n" +
    "- collectionName: The name of the geofence collection (default is
 'AWSLocationCollection')\n" +
    "- geoId: The geographic identifier used for the geofence or map (default is
 'geoId')\n" +
    "- trackerName: The name of the tracker (default is 'geoTracker')\n" +
    "- calculatorName: The name of the route calculator (default is
 'AWSRouteCalc')\n" +
    "- deviceId: The ID of the device (default is 'iPhone-112356')",

  { header: true },
);
const displayCreateAMap = new ScenarioOutput(
  "displayCreateAMap",
  "1. Create a map\n" +
    "An AWS Location map can enhance the user experience of your " +
    " application by providing accurate and personalized location-based " +
    " features. For example, you could use the geocoding capabilities to " +
```

```
    " allow users to search for and locate businesses, landmarks, or " +
    " other points of interest within a specific region.",
);

const sdkCreateAMap = new ScenarioAction(
  "sdkCreateAMap",
  async (/** @type {State} */ state) => {
    const createMapParams = {
      MapName: `${data.inputs.mapName}`,
      Configuration: { style: "VectorEsriNavigation" },
    };
    try {
      const command = new CreateMapCommand(createMapParams);
      const response = await locationClient.send(command);
      state.MapName = response.MapName;
      console.log("Map created. Map ARN is: ", state.MapName);
    } catch (error) {
      console.error("Error creating map: ", error);
      throw error;
    }
  },
);

const displayMapUrl = new ScenarioOutput(
  "displayMapUrl",
  "2. Display Map URL\n" +
    "When you embed a map in a web app or website, the API key is " +
    "included in the map tile URL to authenticate requests. You can " +
    "restrict API keys to specific AWS Location operations (e.g., only " +
    "maps, not geocoding). API keys can expire, ensuring temporary " +
    "access control.\n" +
    "In order to get the MAP URL you need to create and get the API Key value. " +
    "You can create and get the key value using the AWS Management Console under " +
    "Location Services. These operations cannot be completed using the " +
    "AWS SDK. For more information about getting the key value, see " +
    "the AWS Location Documentation.",
);

const sdkDisplayMapUrl = new ScenarioAction(
  "sdkDisplayMapUrl",
  async (/** @type {State} */ state) => {
    const mapURL = `https://maps.geo.aws.amazon.com/maps/v0/maps/${state.MapName}/
tiles/{z}/{x}/{y}?key=API_KEY_VALUE`;
    state.mapURL = mapURL;
```

```
    console.log(
      `Replace \'API_KEY_VALUE\' in the following URL with the value for the API key
 you create and get from the AWS Management Console under Location Services. This is
 then the Map URL you can embed this URL in your Web app:\n
${state.mapURL}`,
    );
  },
);
const displayCreateGeoFenceColl = new ScenarioOutput(
  "displayCreateGeoFenceColl",
  "3. Create a geofence collection, which manages and stores geofences.",
);

const sdkCreateGeoFenceColl = new ScenarioAction(
  "sdkCreateGeoFenceColl",
  async (/** @type {State} */ state) => {
    // Creates a new geofence collection.
    const geoFenceCollParams = {
      CollectionName: `${data.inputs.collectionName}`,
    };
    try {
      const command = new CreateGeofenceCollectionCommand(geoFenceCollParams);
      const response = await locationClient.send(command);
      state.CollectionName = response.CollectionName;
      console.log(
        `The geofence collection was successfully created: ${state.CollectionName}`,
      );
    } catch (caught) {
      if (caught instanceof ConflictException) {
        console.error(
          `An unexpected error occurred while creating the geofence collection:
 ${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
  },
);
const displayStoreGeometry = new ScenarioOutput(
  "displayStoreGeometry",
  "4. Store a geofence geometry in a given geofence collection. " +
    "An AWS Location geofence is a virtual boundary that defines a geographic area "
 +
    "on a map. It is a useful feature for tracking the location of " +
```

```
    "assets or monitoring the movement of objects within a specific region. " +
    "To define a geofence, you need to specify the coordinates of a " +
    "polygon that represents the area of interest. The polygon must be " +
    "defined in a counter-clockwise direction, meaning that the points of " +
    "the polygon must be listed in a counter-clockwise order. " +
    "This is a requirement for the AWS Location service to correctly " +
    "interpret the geofence and ensure that the location data is " +
    "accurately processed within the defined area.",
);

const sdkStoreGeometry = new ScenarioAction(
  "sdkStoreGeometry",
  async (/** @type {State} */ state) => {
    const geoFenceGeoParams = {
      CollectionName: `${data.inputs.collectionName}`,
      GeofenceId: `${data.inputs.geoId}`,
      Geometry: {
        Polygon: [
          [
            [-122.3381, 47.6101],
            [-122.3281, 47.6101],
            [-122.3281, 47.6201],
            [-122.3381, 47.6201],
            [-122.3381, 47.6101],
          ],
        ],
      },
    };
    try {
      const command = new PutGeofenceCommand(geoFenceGeoParams);
      const response = await locationClient.send(command);
      state.GeoFencId = response.GeofenceId;
      console.log("GeoFence created. GeoFence ID is: ", state.GeoFencId);
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
          `A validation error occurred while creating geofence: ${caught.message} \n
 Exiting program.`,
        );
        return;
      }
    }
  },
);
```

```
const displayCreateTracker = new ScenarioOutput(
  "displayCreateTracker",
  "5. Create a tracker resource which lets you retrieve current and historical
 location of devices.",
);

const sdkCreateTracker = new ScenarioAction(
  "sdkCreateTracker",
  async (/** @type {State} */ state) => {
    //Creates a new tracker resource in your AWS account, which you can use to track
 the location of devices.
    const createTrackerParams = {
      TrackerName: `${data.inputs.trackerName}`,
      Description: "Created using the JavaScript V3 SDK",
      PositionFiltering: "TimeBased",
    };
    try {
      const command = new CreateTrackerCommand(createTrackerParams);
      const response = await locationClient.send(command);
      state.trackerName = response.TrackerName;
      console.log("Tracker created. Tracker name is : ", state.trackerName);
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `A validation error occurred while creating geofence: ${caught.message} \n
 Exiting program.`,
        );
      } else {
        `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
      }
      return;
    }
  },
);
const displayUpdatePosition = new ScenarioOutput(
  "displayUpdatePosition",
  "6. Update the position of a device in the location tracking system." +
    "The AWS Location Service does not enforce a strict format for deviceId, but it
 must:\n " +
    "- Be a string (case-sensitive).\n" +
    "- Be 1–100 characters long.\n" +
    "- Contain only: Alphanumeric characters (A-Z, a-z, 0-9); Underscores (_);
 Hyphens (-); and be the same ID used when sending and retrieving positions.",
);
```

```
const sdkUpdatePosition = new ScenarioAction(
  "sdkUpdatePosition",
  async (/** @type {State} */ state) => {
    // Updates the position of a device in the location tracking system.

    const updateDevicePosParams = {
      TrackerName: `${data.inputs.trackerName}`,
      Updates: [
        {
          DeviceId: `${data.inputs.deviceId}`,
          SampleTime: new Date(),
          Position: [-122.4194, 37.7749],
        },
      ],
    };
    try {
      const command = new BatchUpdateDevicePositionCommand(
        updateDevicePosParams,
      );
      const response = await locationClient.send(command);
      console.log(
        `Device with id ${data.inputs.deviceId} was successfully updated in the
 location tracking system. `,
      );
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `A validation error occurred while updating the device: ${caught.message}
 \n Exiting program.`,
        );
      }
    }
  },
);
const displayRetrievePosition = new ScenarioOutput(
  "displayRetrievePosition",
  "7. Retrieve the most recent position update for a specified device.",
);

const sdkRetrievePosition = new ScenarioAction(
  "sdkRetrievePosition",
  async (/** @type {State} */ state) => {
    const devicePositionParams = {
```

```
      TrackerName: `${data.inputs.trackerName}`,
      DeviceId: `${data.inputs.deviceId}`,
    };
    try {
      const command = new GetDevicePositionCommand(devicePositionParams);
      const response = await locationClient.send(command);
      state.position = response.Position;
      console.log("Successfully fetched device position: : ", state.position);
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `"The resource was not found: ${caught.message} \n Exiting program.`,
        );
      } else {
        `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
      }
      return;
    }
  },
);
const displayCreateRouteCalc = new ScenarioOutput(
  "displayCreateRouteCalc",
  "8. Create a route calculator.",
);

const sdkCreateRouteCalc = new ScenarioAction(
  "sdkCreateRouteCalc",
  async (/** @type {State} */ state) => {
    const routeCalcParams = {
      CalculatorName: `${data.inputs.calculatorName}`,
      DataSource: "Esri",
    };
    try {
      // Creates a new route calculator with the specified name and data source.
      const command = new CreateRouteCalculatorCommand(routeCalcParams);
      const response = await locationClient.send(command);
      state.CalculatorName = response.CalculatorName;
      console.log(
        "Route calculator created successfully. Calculator name is: ",
        state.CalculatorName,
      );
    } catch (caught) {
      if (caught instanceof ConflictException) {
        console.error(
```

```
          `An conflict occurred: ${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
  },
);
const displayDetermineDist = new ScenarioOutput(
  "displayDetermineDist",
  "9. Determine the distance between Seattle and Vancouver using the route
 calculator.",
);

const sdkDetermineDist = new ScenarioAction(
  "sdkDetermineDist",
  async (/** @type {State} */ state) => {
    // Calculates the distance between two locations asynchronously.
    const determineDist = {
      CalculatorName: `${data.inputs.calculatorName}`,
      DeparturePosition: [-122.3321, 47.6062],
      DestinationPosition: [-123.1216, 49.2827],
      TravelMode: "Car",
      DistanceUnit: "Kilometers",
    };
    try {
      const command = new CalculateRouteCommand(determineDist);
      const response = await locationClient.send(command);

      console.log(
        "Successfully calculated route. The distance in kilometers is : ",
        response.Summary.Distance,
      );
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `Failed to calculate route: ${caught.message} \n Exiting program.`,
        );
      }
      return;
    }
  },
);
const displayUseGeoPlacesClient = new ScenarioOutput(
  "displayUseGeoPlacesClient",
```

```
  "10. Use the GeoPlacesAsyncClient to perform additional operations. " +
    "This scenario will show use of the GeoPlacesClient that enables" +
    "location search and geocoding capabilities for your applications. " +
    "We are going to use this client to perform these AWS Location tasks: \n" +
    " - Reverse Geocoding (reverseGeocode): Converts geographic coordinates into
 addresses.\n " +
    " - Place Search (searchText): Finds places based on search queries.\n " +
    " - Nearby Search (searchNearby): Finds places near a specific location.\n " +
    "First we will perform a Reverse Geocoding operation",
);

const sdkUseGeoPlacesClient = new ScenarioAction(
  "sdkUseGeoPlacesClient",
  async (/** @type {State} */ state) => {
    const geoPlacesClient = new GeoPlacesClient({ region: region });

    const reverseGeoCodeParams = {
      QueryPosition: [-122.4194, 37.7749],
    };
    const searchTextParams = {
      QueryText: "coffee shop",
      BiasPosition: [-122.4194, 37.7749], //San Fransisco
    };
    const searchNearbyParams = {
      QueryPosition: [-122.4194, 37.7749],
      QueryRadius: Number("1000"),
    };
    try {
      /*   Performs reverse geocoding using the AWS Geo Places API.
     Reverse geocoding is the process of converting geographic coordinates (latitude
 and longitude) to a human-readable address.
     This method uses the latitude and longitude of San Francisco as the input, and
 prints the resulting address.*/

      console.log("Use latitude 37.7749 and longitude -122.4194.");
      const command = new ReverseGeocodeCommand(reverseGeoCodeParams);
      const response = await geoPlacesClient.send(command);
      console.log(
        "Successfully calculated route. The distance in kilometers is : ",
        response.ResultItems[0].Distance,
      );
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
```

```
          `An conflict occurred: ${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
    try {
      console.log(
        "Now we are going to perform a text search using coffee shop",
      );

      /*Searches for a place using the provided search query and prints the detailed
information of the first result.
 @param searchTextParams the search query to be used for the place search (ex,
coffee shop)*/

      const command = new SearchTextCommand(searchTextParams);
      const response = await geoPlacesClient.send(command);
      const placeId = response.ResultItems[0].PlaceId.toString();
      const getPlaceCommand = new GetPlaceCommand({
        PlaceId: placeId,
      });
      const getPlaceResponse = await geoPlacesClient.send(getPlaceCommand);
      console.log(
        `Detailed Place Information: \n Name and address:
${getPlaceResponse.Address.Label}`,
      );

      const foodTypes = getPlaceResponse.FoodTypes;
      if (foodTypes.length) {
        console.log("Food Types: ");
        for (const foodType of foodTypes) {
          console.log("- ", foodType.LocalizedName);
        }
      } else {
        console.log("No food types available.");
      }
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
          `An conflict occurred: ${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
```

```
    try {
      console.log("\nNow we are going to perform a nearby search.");
      const command = new SearchNearbyCommand(searchNearbyParams);
      const response = await geoPlacesClient.send(command);
      const resultItems = response.ResultItems;
      console.log("\nSuccessfully performed nearby search.");
      for (const resultItem of resultItems) {
        console.log("Name and address: ", resultItem.Address.Label);
        console.log("Distance: ", resultItem.Distance);
      }
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
          `An conflict occurred: ${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
  },
);

const displayDeleteResources = new ScenarioOutput(
  "displayDeleteResources",
  "11. Delete the AWS Location Services resources. " +
    "Would you like to delete the AWS Location Services resources? (y/n)",
);

const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    const deleteGeofenceCollParams = {
      CollectionName: `${state.CollectionName}`,
    };
    const deleteRouteCalculatorParams = {
      CalculatorName: `${state.CalculatorName}`,
    };
    const deleteTrackerParams = { TrackerName: `${state.trackerName}` };
    const deleteMapParams = { MapName: `${state.MapName}` };
    try {
      const command = new DeleteMapCommand(deleteMapParams);
      const response = await locationClient.send(command);
      console.log("Map deleted.");
    } catch (error) {
      console.log("Error deleting map: ", error);
```

```
    }
    try {
      const command = new DeleteGeofenceCollectionCommand(
        deleteGeofenceCollParams,
      );
      const response = await locationClient.send(command);
      console.log("Geofence collection deleted.");
    } catch (error) {
      console.log("Error deleting geofence collection: ", error);
    }
    try {
      const command = new DeleteRouteCalculatorCommand(
        deleteRouteCalculatorParams,
      );
      const response = await locationClient.send(command);
      console.log("Route calculator deleted.");
    } catch (error) {
      console.log("Error deleting route calculator: ", error);
    }
    try {
      const command = new DeleteTrackerCommand(deleteTrackerParams);
      const response = await locationClient.send(command);
      console.log("Tracker deleted.");
    } catch (error) {
      console.log("Error deleting tracker: ", error);
    }
  },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "Thank you for checking out the Amazon Location Service Use demo. We hope you " +
    "learned something new, or got some inspiration for your own apps today!" +
    " For more Amazon Location Services examples in different programming languages,
 have a look at: " +
    "https://docs.aws.amazon.com/code-library/latest/ug/
location_code_examples.html",
);

const myScenario = new Scenario("Location Services Scenario", [
  greet,
  pressEnter,
  displayCreateAMap,
  sdkCreateAMap,
```

```
  pressEnter,
  displayMapUrl,
  sdkDisplayMapUrl,
  pressEnter,
  displayCreateGeoFenceColl,
  sdkCreateGeoFenceColl,
  pressEnter,
  displayStoreGeometry,
  sdkStoreGeometry,
  pressEnter,
  displayCreateTracker,
  sdkCreateTracker,
  pressEnter,
  displayUpdatePosition,
  sdkUpdatePosition,
  pressEnter,
  displayRetrievePosition,
  sdkRetrievePosition,
  pressEnter,
  displayCreateRouteCalc,
  sdkCreateRouteCalc,
  pressEnter,
  displayDetermineDist,
  sdkDetermineDist,
  pressEnter,
  displayUseGeoPlacesClient,
  sdkUseGeoPlacesClient,
  pressEnter,
  displayDeleteResources,
  pressEnterConfirm,
  sdkDeleteResources,
  goodbye,
]);

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

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

```
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

Actions

`BatchUpdateDevicePosition`

L'exemple de code suivant montre comment utiliser`BatchUpdateDevicePosition`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  BatchUpdateDevicePositionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };
const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });
const updateDevicePosParams = {
  TrackerName: `${data.inputs.trackerName}`,
  Updates: [
    {
      DeviceId: `${data.inputs.deviceId}`,
      SampleTime: new Date(),
      Position: [-122.4194, 37.7749],
    },
  ],
```

```
};
export const main = async () => {
  try {
    const command = new BatchUpdateDevicePositionCommand(updateDevicePosParams);
    const response = await locationClient.send(command);
    //console.log("response ", response.Errors[0].Error);

    console.log(
      `Device with id ${data.inputs.deviceId} was successfully updated in the
 location tracking system. `,
      response,
    );
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous BatchUpdateDevicePositionà la section Référence des AWS SDK pour JavaScript API.

`CalculateRoute`

L'exemple de code suivant montre comment utiliser`CalculateRoute`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  CalculateRouteCommand,
  ResourceNotFoundException,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };
```

```
const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });

export const main = async () => {
  const routeCalcParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
    DeparturePosition: [-122.3321, 47.6062],
    DestinationPosition: [-123.1216, 49.2827],
    TravelMode: "Car",
    DistanceUnit: "Kilometers",
  };
  try {
    const command = new CalculateRouteCommand(routeCalcParams);
    const response = await locationClient.send(command);

    console.log(
      "Successfully calculated route. The distance in kilometers is : ",
      response.Summary.Distance,
    );
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `An conflict occurred: ${caught.message} \n Exiting program.`,
      );
      return;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CalculateRouteà la section Référence des AWS SDK pour JavaScript API.

`CreateGeofenceCollection`

L'exemple de code suivant montre comment utiliser`CreateGeofenceCollection`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  ConflictException,
  CreateGeofenceCollectionCommand,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const geoFenceCollParams = {
    CollectionName: `${data.inputs.collectionName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateGeofenceCollectionCommand(geoFenceCollParams);
    const response = await locationClient.send(command);
    console.log(
      "Collection created. Collection name is: ",
      response.CollectionName,
    );
  } catch (caught) {
    if (caught instanceof ConflictException) {
      console.error("A conflict occurred. Exiting program.");
      return;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateGeofenceCollectionà la section Référence des AWS SDK pour JavaScript API.

`CreateMap`

L'exemple de code suivant montre comment utiliser`CreateMap`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import { CreateMapCommand, LocationClient } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const CreateMapCommandInput = {
    MapName: `${data.inputs.mapName}`,
    Configuration: { style: "VectorEsriNavigation" },
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateMapCommand(CreateMapCommandInput);
    const response = await locationClient.send(command);
    console.log("Map created. Map ARN is : ", response.MapArn);
  } catch (error) {
    console.error("Error creating map: ", error);
    throw error;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateMapà la section Référence des AWS SDK pour JavaScript API.

`CreateRouteCalculator`

L'exemple de code suivant montre comment utiliser`CreateRouteCalculator`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  ConflictException,
  CreateRouteCalculatorCommand,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });

export const main = async () => {
  const routeCalcParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
    DataSource: "Esri",
  };
  try {
    const command = new CreateRouteCalculatorCommand(routeCalcParams);
    const response = await locationClient.send(command);

    console.log(
      "Route calculator created successfully. Calculator name is ",
      response.CalculatorName,
    );
  } catch (caught) {
    if (caught instanceof ConflictException) {
      console.error(
        `An conflict occurred: ${caught.message} \n Exiting program.`,
      );
      return;
    }
```

```
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateRouteCalculatorà la section Référence des AWS SDK pour JavaScript API.

`CreateTracker`

L'exemple de code suivant montre comment utiliser`CreateTracker`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import { CreateTrackerCommand, LocationClient } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const createTrackerParams = {
    TrackerName: `${data.inputs.trackerName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateTrackerCommand(createTrackerParams);
    const response = await locationClient.send(command);
    //state.trackerName - response.TrackerName;
    console.log("Tracker created. Tracker name is : ", response.TrackerName);
  } catch (error) {
    console.error("Error creating map: ", error);
    throw error;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateTrackerà la section Référence des AWS SDK pour JavaScript API.

`DeleteGeofenceCollection`

L'exemple de code suivant montre comment utiliser`DeleteGeofenceCollection`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  DeleteGeofenceCollectionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteGeofenceCollParams = {
    CollectionName: `${data.inputs.collectionName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new DeleteGeofenceCollectionCommand(
      deleteGeofenceCollParams,
    );
    const response = await locationClient.send(command);
    console.log("Collection deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
```

```
      console.error(
        `${data.inputs.collectionName} Geofence collection not found.`,
      );
      return;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteGeofenceCollectionà la section Référence des AWS SDK pour JavaScript API.

`DeleteMap`

L'exemple de code suivant montre comment utiliser`DeleteMap`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  DeleteMapCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteMapParams = {
    MapName: `${data.inputs.mapName}`,
  };
  try {
    const locationClient = new LocationClient({ region: region });
```

```
    const command = new DeleteMapCommand(deleteMapParams);
    const response = await locationClient.send(command);
    console.log("Map deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(`${data.inputs.mapName} map not found.`);
      return;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteMapà la section Référence des AWS SDK pour JavaScript API.

`DeleteRouteCalculator`

L'exemple de code suivant montre comment utiliser`DeleteRouteCalculator`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  DeleteRouteCalculatorCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteRouteCalculatorParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
```

```
  };
  try {
    const locationClient = new LocationClient({ region: region });
    const command = new DeleteRouteCalculatorCommand(
      deleteRouteCalculatorParams,
    );
    const response = await locationClient.send(command);
    console.log("Route calculator deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `${data.inputs.calculatorName} route calculator not found.`,
      );
      return;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteRouteCalculatorà la section Référence des AWS SDK pour JavaScript API.

`DeleteTracker`

L'exemple de code suivant montre comment utiliser`DeleteTracker`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  DeleteTrackerCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
```

```
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteTrackerParams = {
    TrackerName: `${data.inputs.trackerName}`,
  };
  try {
    const locationClient = new LocationClient({ region: region });
    const command = new DeleteTrackerCommand(deleteTrackerParams);
    const response = await locationClient.send(command);
    console.log("Tracker deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(`${data.inputs.trackerName} tracker not found.`);
      return;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteTrackerà la section Référence des AWS SDK pour JavaScript API.

`GetDevicePosition`

L'exemple de code suivant montre comment utiliser`GetDevicePosition`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

```
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const locationClient = new LocationClient({ region: region });
  const deviceId = `${data.inputs.deviceId}`;
  const trackerName = `${data.inputs.trackerName}`;

  const devicePositionParams = {
    DeviceId: deviceId,
    TrackerName: trackerName,
  };
  try {
    const command = new GetDevicePositionCommand(devicePositionParams);
    const response = await locationClient.send(command);
    //state.position = response.position;
    console.log("Successfully fetched device position: ", response);
  } catch (error) {
    console.log("Error ", error);
    /*  if (caught instanceof ResourceNotFoundException) {
      console.error(
        `"The resource was not found: ${caught.message} \n Exiting program.`,
      );
    } else {
      `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
    }
    return;*/
  }
};
```

- Pour plus de détails sur l'API, reportez-vous GetDevicePositionà la section Référence des AWS SDK pour JavaScript API.

`PutGeofence`

L'exemple de code suivant montre comment utiliser`PutGeofence`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { fileURLToPath } from "node:url";
import {
  PutGeofenceCommand,
  LocationClient,
  ValidationException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });
export const main = async () => {
  const geoFenceGeoParams = {
    CollectionName: `${data.inputs.collectionName}`,
    GeofenceId: `${data.inputs.geoId}`,
    Geometry: {
      Polygon: [
        [
          [-122.3381, 47.6101],
          [-122.3281, 47.6101],
          [-122.3281, 47.6201],
          [-122.3381, 47.6201],
          [-122.3381, 47.6101],
        ],
      ],
    },
  };
  try {
    const command = new PutGeofenceCommand(geoFenceGeoParams);
    const response = await locationClient.send(command);
    console.log("GeoFence created. GeoFence ID is: ", response.GeofenceId);
  } catch (error) {
    console.error(
      `A validation error occurred while creating geofence: ${error} \n Exiting
 program.`,
```

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

- Pour plus de détails sur l'API, reportez-vous PutGeofenceà la section Référence des AWS SDK pour JavaScript API.

Exemples Amazon MSK utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon MSK.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Exemples sans serveur

Exemples sans serveur

Invocation d'une fonction Lambda à partir d'un déclencheur Amazon MSK

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un cluster Amazon MSK. La fonction récupère les données utiles MSK et journalise le contenu de l'enregistrement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Amazon MSK avec JavaScript Lambda à l'aide de.

```
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)
      })
    }
}
```

Utilisation d'un événement Amazon MSK avec TypeScript Lambda à l'aide de.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
```

```
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    };
  }
}
```

Exemples Amazon Personalize à l'aide du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Personalize.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`CreateBatchInferenceJob`

L'exemple de code suivant montre comment utiliser`CreateBatchInferenceJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchInferenceJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch inference job's parameters.

export const createBatchInferenceJobParam = {
  jobName: "JOB_NAME",
  jobInput: {
    s3DataSource: {
      path: "INPUT_PATH",
    },
  },
  jobOutput: {
    s3DataDestination: {
      path: "OUTPUT_PATH",
    },
  },
  roleArn: "ROLE_ARN",
  solutionVersionArn: "SOLUTION_VERSION_ARN",
  numResults: 20,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateBatchInferenceJobCommand(createBatchInferenceJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateBatchInferenceJobà la section Référence des AWS SDK pour JavaScript API.

`CreateBatchSegmentJob`

L'exemple de code suivant montre comment utiliser`CreateBatchSegmentJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: "NAME",
  jobInput: {
    s3DataSource: {
      path: "INPUT_PATH",
    },
  },
  jobOutput: {
    s3DataDestination: {
      path: "OUTPUT_PATH",
    },
  },
  roleArn: "ROLE_ARN",
  solutionVersionArn: "SOLUTION_VERSION_ARN",
  numResults: 20,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateBatchSegmentJobCommand(createBatchSegmentJobParam),
    );
```

```
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateBatchSegmentJobà la section Référence des AWS SDK pour JavaScript API.

`CreateCampaign`

L'exemple de code suivant montre comment utiliser`CreateCampaign`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.

import { CreateCampaignCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the campaign's parameters.
export const createCampaignParam = {
  solutionVersionArn: "SOLUTION_VERSION_ARN" /* required */,
  name: "NAME" /* required */,
  minProvisionedTPS: 1 /* optional integer */,
};

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

```
    const response = await personalizeClient.send(
      new CreateCampaignCommand(createCampaignParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateCampaignà la section Référence des AWS SDK pour JavaScript API.

`CreateDataset`

L'exemple de code suivant montre comment utiliser`CreateDataset`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset's parameters.
export const createDatasetParam = {
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  datasetType: "DATASET_TYPE" /* required */,
  name: "NAME" /* required */,
  schemaArn: "SCHEMA_ARN" /* required */,
};
```

```
export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetCommand(createDatasetParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateDatasetà la section Référence des AWS SDK pour JavaScript API.

`CreateDatasetExportJob`

L'exemple de code suivant montre comment utiliser`CreateDatasetExportJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetExportJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the export job parameters.
export const datasetExportJobParam = {
  datasetArn: "DATASET_ARN" /* required */,
  jobOutput: {
```

```
    s3DataDestination: {
      path: "S3_DESTINATION_PATH" /* required */,
      //kmsKeyArn: 'ARN'  /* include if your bucket uses AWS KMS for encryption
    },
  },
  jobName: "NAME" /* required */,
  roleArn: "ROLE_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetExportJobCommand(datasetExportJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateDatasetExportJobà la section Référence des AWS SDK pour JavaScript API.

`CreateDatasetGroup`

L'exemple de code suivant montre comment utiliser`CreateDatasetGroup`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.

import { CreateDatasetGroupCommand } from "@aws-sdk/client-personalize";
```

```
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset group parameters.
export const createDatasetGroupParam = {
  name: "NAME" /* required */,
};

export const run = async (createDatasetGroupParam) => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetGroupCommand(createDatasetGroupParam),
    );
    console.log("Success", response);
    return "Run successfully"; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run(createDatasetGroupParam);
```

Créez un groupe de jeu de données de domaine.

```
// 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();
```

- Pour plus de détails sur l'API, reportez-vous CreateDatasetGroupà la section Référence des AWS SDK pour JavaScript API.

`CreateDatasetImportJob`

L'exemple de code suivant montre comment utiliser`CreateDatasetImportJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetImportJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset import job parameters.
export const datasetImportJobParam = {
  datasetArn: "DATASET_ARN" /* required */,
  dataSource: {
    /* required */
    dataLocation: "S3_PATH",
  },
  jobName: "NAME" /* required */,
```

```
  roleArn: "ROLE_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetImportJobCommand(datasetImportJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateDatasetImportJobà la section Référence des AWS SDK pour JavaScript API.

`CreateEventTracker`

L'exemple de code suivant montre comment utiliser`CreateEventTracker`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateEventTrackerCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the event tracker's parameters.
export const createEventTrackerParam = {
```

```
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  name: "NAME" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateEventTrackerCommand(createEventTrackerParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateEventTrackerà la section Référence des AWS SDK pour JavaScript API.

`CreateFilter`

L'exemple de code suivant montre comment utiliser`CreateFilter`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateFilterCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the filter's parameters.
export const createFilterParam = {
```

```
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  name: "NAME" /* required */,
  filterExpression: "FILTER_EXPRESSION" /*required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateFilterCommand(createFilterParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateFilterà la section Référence des AWS SDK pour JavaScript API.

`CreateRecommender`

L'exemple de code suivant montre comment utiliser`CreateRecommender`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateRecommenderCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});
```

```
// Set the recommender's parameters.
export const createRecommenderParam = {
  name: "NAME" /* required */,
  recipeArn: "RECIPE_ARN" /* required */,
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateRecommenderCommand(createRecommenderParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateRecommenderà la section Référence des AWS SDK pour JavaScript API.

`CreateSchema`

L'exemple de code suivant montre comment utiliser`CreateSchema`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
```

```
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from "node:fs";

const schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = "TEST"; // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: "NAME" /* required */,
  schema: mySchema /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSchemaCommand(createSchemaParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Créez un schéma avec un domaine.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from "node:fs";
```

```
const schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = "TEST"; // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: "NAME" /* required */,
  schema: mySchema /* required */,
  domain:
    "DOMAIN" /* required for a domain dataset group, specify ECOMMERCE or
 VIDEO_ON_DEMAND */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSchemaCommand(createDomainSchemaParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateSchemaà la section Référence des AWS SDK pour JavaScript API.

`CreateSolution`

L'exemple de code suivant montre comment utiliser`CreateSolution`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution parameters.
export const createSolutionParam = {
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  recipeArn: "RECIPE_ARN" /* required */,
  name: "NAME" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSolutionCommand(createSolutionParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateSolutionà la section Référence des AWS SDK pour JavaScript API.

`CreateSolutionVersion`

L'exemple de code suivant montre comment utiliser`CreateSolutionVersion`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionVersionCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution version parameters.
export const solutionVersionParam = {
  solutionArn: "SOLUTION_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSolutionVersionCommand(solutionVersionParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateSolutionVersionà la section Référence des AWS SDK pour JavaScript API.

Exemples d'Amazon Personalize Events à l'aide du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Personalize Events.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Actions

`PutEvents`

L'exemple de code suivant montre comment utiliser`PutEvents`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
```

```
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
 Replace it with your sentAt timestamp in UNIX format.

// Set put events parameters.
const putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous PutEventsà la section Référence des AWS SDK pour JavaScript API.

`PutItems`

L'exemple de code suivant montre comment utiliser`PutItems`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { PutItemsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put items parameters. For string properties and values, use the \
 character to escape quotes.
const putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /*  required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
 "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous PutItemsà la section Référence des AWS SDK pour JavaScript API.

`PutUsers`

L'exemple de code suivant montre comment utiliser`PutUsers`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { PutUsersCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put users parameters. For string properties and values, use the \
 character to escape quotes.
const putUsersParam = {
  datasetArn: "DATASET_ARN",
  users: [
    {
      userId: "USER_ID",
      properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
```

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

- Pour plus de détails sur l'API, reportez-vous PutUsersà la section Référence des AWS SDK pour JavaScript API.

Exemples d'Amazon Personalize Runtime à l'aide du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Personalize Runtime.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`GetPersonalizedRanking`

L'exemple de code suivant montre comment utiliser`GetPersonalizedRanking`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { GetPersonalizedRankingCommand } from "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the ranking request parameters.
export const getPersonalizedRankingParam = {
  campaignArn: "CAMPAIGN_ARN" /* required */,
  userId: "USER_ID" /* required */,
  inputList: ["ITEM_ID_1", "ITEM_ID_2", "ITEM_ID_3", "ITEM_ID_4"],
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetPersonalizedRankingCommand(getPersonalizedRankingParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous GetPersonalizedRankingà la section Référence des AWS SDK pour JavaScript API.

`GetRecommendations`

L'exemple de code suivant montre comment utiliser`GetRecommendations`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from "@aws-sdk/client-personalize-runtime";

import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: "CAMPAIGN_ARN" /* required */,
  userId: "USER_ID" /* required */,
  numResults: 15 /* optional */,
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetRecommendationsCommand(getRecommendationsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Obtenez des recommandations à l'aide d'un filtre (groupe de jeu de données personnalisé).

```
// 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();
```

Obtenez des recommandations filtrées à partir d'un système de recommandation créé dans un groupe de jeux de données de domaine.

```
// 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();
```

- Pour plus de détails sur l'API, reportez-vous GetRecommendationsà la section Référence des AWS SDK pour JavaScript API.

Exemples d'Amazon Pinpoint utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Pinpoint.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`SendMessages`

L'exemple de code suivant montre comment utiliser`SendMessages`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
export const pinClient = new PinpointClient({ region: REGION });
```

Envoyer un e-mail.

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

// The FromAddress must be verified in SES.
const fromAddress = "FROM_ADDRESS";
const toAddress = "TO_ADDRESS";
const projectId = "PINPOINT_PROJECT_ID";

// The subject line of the email.
const subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
const body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;
```

```
// The body of the email for recipients whose email clients support HTML content.
const body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
 using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding for the subject line and message body of the email.
const charset = "UTF-8";

const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
```

```
  },
};

const run = async () => {
  try {
    const { MessageResponse } = await pinClient.send(
      new SendMessagesCommand(params),
    );

    if (!MessageResponse) {
      throw new Error("No message response.");
    }

    if (!MessageResponse.Result) {
      throw new Error("No message result.");
    }

    const recipientResult = MessageResponse.Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    }
    console.log(recipientResult.MessageId);
  } catch (err) {
    console.log(err.message);
  }
};

run();
```

Envoyer un SMS.

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

/* The phone number or short code to send the message from. The phone number
 or short code that you specify has to be associated with your Amazon Pinpoint
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX
```

```
// The recipient's phone number.  For best results, you should specify the phone
 number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
const messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
const registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

const senderId = "MySenderID";

// Specify the parameters to pass to the API.
const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
```

```
        SenderId: senderId,
      },
    },
  },
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    console.log(
      `Message sent!
 ${data.MessageResponse.Result[destinationNumber].StatusMessage}`,
    );
  } catch (err) {
    console.log(err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous SendMessagesà la section Référence des AWS SDK pour JavaScript API.

Exemples d'Amazon Polly utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Polly.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Scénarios

Scénarios

Créez une application pour analyser les commentaires des clients

L'exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour JavaScript (v3)

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub. Les extraits suivants montrent comment le AWS SDK pour JavaScript est utilisé dans les fonctions Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
```

```
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
```

```
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
```

```
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda

- Amazon Polly
- Amazon Textract
- Amazon Translate

Exemples d'Amazon RDS utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon RDS.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Scénarios

Créer un outil de suivi des éléments de travail sans serveur Aurora

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une base de données Amazon Aurora sans serveur et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK pour JavaScript (v3) pour créer une application Web qui suit les éléments de travail dans une base de données Amazon Aurora et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise un front end créé avec React.js pour interagir avec un backend Express Node.js.

- Intégrez une application Web React.js à Services AWS.
- Lister, ajouter et mettre à jour des éléments dans une table Aurora.
- Envoyez un rapport par e-mail sur les éléments de travail filtrés en utilisant Amazon SES.

- Déployez et gérez des exemples de ressources à l'aide du AWS CloudFormation script inclus.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Aurora
- Amazon RDS
- Services de données Amazon RDS
- Amazon SES

Exemples sans serveur

Connexion à une base de données Amazon RDS dans une fonction Lambda

L'exemple de code suivant montre comment implémenter une fonction Lambda qui se connecte à une base de données RDS. La fonction effectue une simple requête de base de données et renvoie le résultat.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une base de données Amazon RDS dans une fonction Lambda à l'aide de. 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)
  }
};
```

Connexion à une base de données Amazon RDS dans une fonction Lambda à l'aide de. 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}`)
    };
};
```

Exemples d'Amazon RDS Data Service utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec Amazon RDS Data Service.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Scénarios

Créer un outil de suivi des éléments de travail sans serveur Aurora

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une base de données Amazon Aurora sans serveur et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK pour JavaScript (v3) pour créer une application Web qui suit les éléments de travail dans une base de données Amazon Aurora et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise un front end créé avec React.js pour interagir avec un backend Express Node.js.

- Intégrez une application Web React.js à Services AWS.
- Lister, ajouter et mettre à jour des éléments dans une table Aurora.
- Envoyez un rapport par e-mail sur les éléments de travail filtrés en utilisant Amazon SES.
- Déployez et gérez des exemples de ressources à l'aide du AWS CloudFormation script inclus.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Aurora
- Amazon RDS
- Services de données Amazon RDS
- Amazon SES

Exemples d'Amazon Redshift utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Redshift.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Actions

`CreateCluster`

L'exemple de code suivant montre comment utiliser`CreateCluster`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Créez le cluster.

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

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least
 one uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
 your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
 cluster subnet group to be associated with this cluster. Defaults to 'default' if
 not specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      `Cluster ${data.Cluster.ClusterIdentifier} successfully created`,
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, reportez-vous CreateClusterà la section Référence des AWS SDK pour JavaScript API.

`DeleteCluster`

L'exemple de code suivant montre comment utiliser`DeleteCluster`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Créez le cluster.

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

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

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

- Pour plus de détails sur l'API, reportez-vous DeleteClusterà la section Référence des AWS SDK pour JavaScript API.

`DescribeClusters`

L'exemple de code suivant montre comment utiliser`DescribeClusters`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Décrivez vos clusters.

```
// 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();
```

- Pour plus de détails sur l'API, reportez-vous DescribeClustersà la section Référence des AWS SDK pour JavaScript API.

`ModifyCluster`

L'exemple de code suivant montre comment utiliser`ModifyCluster`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Modifiez un cluster.

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

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
```

```
};

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

- Pour plus de détails sur l'API, reportez-vous ModifyClusterà la section Référence des AWS SDK pour JavaScript API.

Exemples d'Amazon Rekognition utilisant le SDK pour (v3) JavaScript

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Rekognition.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Scénarios

Scénarios

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour JavaScript (v3)

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Détecter des objets dans des images

L'exemple de code suivant montre comment créer une application qui utilise Amazon Rekognition afin de détecter des objets par catégorie dans des images.

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Rekognition AWS SDK pour JavaScript pour créer une application qui utilise Amazon Rekognition pour identifier les objets par catégorie dans des images situées dans un compartiment Amazon Simple Storage Service (Amazon S3). L'application envoie à l'administrateur une notification par e-mail contenant les résultats à l'aide d'Amazon Simple Email Service (Amazon SES).

Découvrez comment :

- Créer un utilisateur non authentifié à l'aide d'Amazon Cognito.
- Analyser les images à la recherche d'objets à l'aide d'Amazon Rekognition.
- Vérifier une adresse e-mail pour Amazon SES.

- Envoyer une notification par e-mail à l'aide d'Amazon SES.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Rekognition
- Amazon S3
- Amazon SES

Exemples d'Amazon S3 utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec Amazon S3.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Hello Amazon S3

L'exemple de code suivant montre comment commencer à utiliser Amazon S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  paginateListBuckets,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * List the S3 buckets in your configured AWS account.
 */
export const helloS3 = async () => {
  // When no region or credentials are provided, the SDK will use the
  // region and credentials from the local AWS config.
  const client = new S3Client({});

  try {
    /**
     * @type { import("@aws-sdk/client-s3").Bucket[] }
     */
    const buckets = [];

    for await (const page of paginateListBuckets({ client }, {})) {
      buckets.push(...page.Buckets);
    }
    console.log("Buckets: ");
    console.log(buckets.map((bucket) => bucket.Name).join("\n"));
    return buckets;
  } catch (caught) {
    // ListBuckets does not throw any modeled errors. Any error caught
```

```
    // here will be something generic like `AccessDenied`.
    if (caught instanceof S3ServiceException) {
      console.error(`${caught.name}: ${caught.message}`);
    } else {
      // Something besides S3 failed.
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ListBucketsà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- créer un compartiment et y charger un fichier ;
- télécharger un objet à partir d'un compartiment ;
- copier un objet dans le sous-dossier d'un compartiment ;
- répertorier les objets d'un compartiment ;
- supprimer le compartiment et tous les objets qui y figurent.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Tout d'abord, importez tous les modules nécessaires.

```
// Used to check if currently running file is this file.
import { fileURLToPath } from "node:url";
import { readdirSync, readFileSync, writeFileSync } from "node:fs";
```

```
// Local helper utils.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

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

Les importations précédentes font référence à certains utilitaires d'annotations. Ces utilitaires sont locaux au GitHub référentiel lié au début de cette section. À titre de référence, consultez les implémentations suivantes de ces utilitaires.

```
export const dirnameFromMetaUrl = (metaUrl) =>
  fileURLToPath(new URL(".", metaUrl));

import { select, input, confirm, checkbox, password } from "@inquirer/prompts";

export class Prompter {
  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  select(options) {
    return select(options);
  }

  /**
   * @param {{ message: string }} options
   */
  input(options) {
    return input(options);
  }

  /**
```

```
   * @param {{ message: string }} options
   */
  password(options) {
    return password({ ...options, mask: true });
  }

  /**
   * @param {string} prompt
   */
  checkContinue = async (prompt = "") => {
    const prefix = prompt && `${prompt} `;
    const ok = await this.confirm({
      message: `${prefix}Continue?`,
    });
    if (!ok) throw new Error("Exiting...");
  };

  /**
   * @param {{ message: string }} options
   */
  confirm(options) {
    return confirm(options);
  }

  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  checkbox(options) {
    return checkbox(options);
  }
}

export const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};
```

Les objets sont stockés dans des « compartiments ». Définissons une fonction pour créer un nouveau compartiment.

```
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;
};
```

Les compartiments contiennent des « objets ». Cette fonction charge le contenu d'un répertoire dans votre compartiment sous forme d'objets.

```
export const uploadFilesToBucket = async ({ bucketName, folderPath }) => {
  console.log(`Uploading files from ${folderPath}\n`);
  const keys = readdirSync(folderPath);
  const files = keys.map((key) => {
    const filePath = `${folderPath}/${key}`;
    const fileContent = readFileSync(filePath);
    return {
      Key: key,
      Body: fileContent,
    };
  });

  for (const file of files) {
    await s3Client.send(
      new PutObjectCommand({
        Bucket: bucketName,
        Body: file.Body,
        Key: file.Key,
      }),
    );
    console.log(`${file.Key} uploaded successfully.`);
  }
};
```

Après avoir chargé des objets, vérifiez qu'ils ont été chargés correctement. Tu peux utiliser ListObjects pour ça. Vous utiliserez la propriété « Key » (Clé), mais la réponse contient également d'autres propriétés utiles.

```
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`);
};
```

Il se peut que vous souhaitiez copier un objet d'un compartiment à un autre. Utilisez la CopyObject commande pour cela.

```
export const copyFileFromBucket = async ({ destinationBucket }) => {
  const proceed = await prompter.confirm({
    message: "Would you like to copy an object from another bucket?",
  });

  if (!proceed) {
    return;
  }
  const copy = async () => {
    try {
      const sourceBucket = await prompter.input({
        message: "Enter source bucket name:",
      });
      const sourceKey = await prompter.input({
        message: "Enter source key:",
      });
      const destinationKey = await prompter.input({
        message: "Enter destination key:",
      });

      const command = new CopyObjectCommand({
        Bucket: destinationBucket,
        CopySource: `${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });
      await s3Client.send(command);
      await copyFileFromBucket({ destinationBucket });
    } catch (err) {
      console.error("Copy error.");
      console.error(err);
      const retryAnswer = await prompter.confirm({ message: "Try again?" });
      if (retryAnswer) {
```

```
        await copy();
      }
    }
  };
  await copy();
};
```

Il n'existe aucune méthode de kit SDK pour obtenir plusieurs objets d'un compartiment. Vous allez plutôt créer une liste d'objets à charger et sur laquelle itérer.

```
export const downloadFilesFromBucket = async ({ bucketName }) => {
  const { Contents } = await s3Client.send(
    new ListObjectsCommand({ Bucket: bucketName }),
  );
  const path = await prompter.input({
    message: "Enter destination path for files:",
  });

  for (const content of Contents) {
    const obj = await s3Client.send(
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
    );
    writeFileSync(
      `${path}/${content.Key}`,
      await obj.Body.transformToByteArray(),
    );
  }
  console.log("Files downloaded successfully.\n");
};
```

Il est temps de nettoyer vos ressources. Un compartiment doit être vide avant de pouvoir être supprimé. Ces deux fonctions vident et suppriment le compartiment.

```
export const emptyBucket = async ({ bucketName }) => {
  const listObjectsCommand = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(listObjectsCommand);
  const keys = Contents.map((c) => c.Key);

  const deleteObjectsCommand = new DeleteObjectsCommand({
    Bucket: bucketName,
```

```
    Delete: { Objects: keys.map((key) => ({ Key: key })) },
  });
  await s3Client.send(deleteObjectsCommand);
  console.log(`${bucketName} emptied successfully.\n`);
};

export const deleteBucket = async ({ bucketName }) => {
  const command = new DeleteBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log(`${bucketName} deleted successfully.\n`);
};
```

La fonction « main » regroupe tout. Si vous exécutez ce fichier directement, la fonction « main » sera appelée.

```
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);
  }
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

Actions

`CopyObject`

L'exemple de code suivant montre comment utiliser`CopyObject`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Copiez l'objet.

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

/**
 * Copy an S3 object from one bucket to another.
 *
 * @param {{
 *   sourceBucket: string,
 *   sourceKey: string,
 *   destinationBucket: string,
 *   destinationKey: string }} config
 */
export const main = async ({
  sourceBucket,
  sourceKey,
  destinationBucket,
  destinationKey,
}) => {
  const client = new S3Client({});

  try {
    await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucket}/${sourceKey}`,
        Bucket: destinationBucket,
        Key: destinationKey,
      }),
    );
    await waitUntilObjectExists(
      { client },
      { Bucket: destinationBucket, Key: destinationKey },
    );
    console.log(
      `Successfully copied ${sourceBucket}/${sourceKey} to ${destinationBucket}/
${destinationKey}`,
    );
  } catch (caught) {
    if (caught instanceof ObjectNotInActiveTierError) {
      console.error(
```

```
        `Could not copy ${sourceKey} from ${sourceBucket}. Object is not in the
 active tier.`,
      );
    } else {
      throw caught;
    }
  }
};
```

Copiez l'objet à condition ETag qu'il ne corresponde pas à celui fourni.

```
import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 string, eTag: string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
  eTag,
}) => {
  const client = new S3Client({});
  const name = data.name;
  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucketName}/${sourceKeyName}`,
        Bucket: destinationBucketName,
```

```
        Key: `${name}${sourceKeyName}`,
        CopySourceIfMatch: eTag,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
 "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Unable to copy object "${sourceKeyName}" to bucket "${sourceBucketName}":
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
    eTag: {
```

```
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Copiez l'objet à condition ETag qu'il ne corresponde pas à celui fourni.

```
import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 string, eTag: string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
```

```
  eTag,
}) => {
  const client = new S3Client({});
  const name = data.name;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucketName}/${sourceKeyName}`,
        Bucket: destinationBucketName,
        Key: `${name}${sourceKeyName}`,
        CopySourceIfNoneMatch: eTag,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
 "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Unable to copy object "${sourceKeyName}" to bucket "${sourceBucketName}":
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
```

```
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Copiez l’objet à condition qu’il ait été créé ou modifié dans un laps de temps donné.

```
import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};
```

```
/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
}) => {
  const date = new Date();
  date.setDate(date.getDate() - 1);

  const name = data.name;
  const client = new S3Client({});
  const copySource = `${sourceBucketName}/${sourceKeyName}`;
  const copiedKey = name + sourceKeyName;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: copySource,
        Bucket: destinationBucketName,
        Key: copiedKey,
        CopySourceIfModifiedSince: date,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
 "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while copying object from ${sourceBucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
```

```
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Copiez l’objet à condition qu’il n’ait pas été créé ou modifié dans un laps de temps donné.

```
import {
```

```
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
}) => {
  const date = new Date();
  date.setDate(date.getDate() - 1);
  const client = new S3Client({});
  const name = data.name;
  const copiedKey = name + sourceKeyName;
  const copySource = `${sourceBucketName}/${sourceKeyName}`;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: copySource,
        Bucket: destinationBucketName,
        Key: copiedKey,
        CopySourceIfUnmodifiedSince: date,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
 "${sourceBucketName}". No such key exists.`,
      );
```

```
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while copying object from ${sourceBucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
```

```
  }
}
```

- Pour plus de détails sur l'API, reportez-vous CopyObjectà la section Référence des AWS SDK pour JavaScript API.

`CreateBucket`

L'exemple de code suivant montre comment utiliser`CreateBucket`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créer le compartiment.

```
import {
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  CreateBucketCommand,
  S3Client,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

/**
 * Create an Amazon S3 bucket.
 * @param {{ bucketName: string }} config
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { Location } = await client.send(
      new CreateBucketCommand({
        // The name of the bucket. Bucket names are unique and have several other
 constraints.
```

```
        // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
bucketnamingrules.html
        Bucket: bucketName,
      }),
    );
    await waitUntilBucketExists({ client }, { Bucket: bucketName });
    console.log(`Bucket created with location ${Location}`);
  } catch (caught) {
    if (caught instanceof BucketAlreadyExists) {
      console.error(
        `The bucket "${bucketName}" already exists in another AWS account. Bucket
 names must be globally unique.`,
      );
    }
    // WARNING: If you try to create a bucket in the North Virginia region,
    // and you already own a bucket in that region with the same name, this
    // error will not be thrown. Instead, the call will return successfully
    // and the ACL on that bucket will be reset.
    else if (caught instanceof BucketAlreadyOwnedByYou) {
      console.error(
        `The bucket "${bucketName}" already exists in this AWS account.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateBucketà la section Référence des AWS SDK pour JavaScript API.

`DeleteBucket`

L'exemple de code suivant montre comment utiliser`DeleteBucket`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez le compartiment.

```
import {
  DeleteBucketCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Delete an Amazon S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new DeleteBucketCommand({
    Bucket: bucketName,
  });

  try {
    await client.send(command);
    console.log("Bucket was deleted.");
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting bucket. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting the bucket. ${caught.name}:
 ${caught.message}`,
      );
    } else {
```

```
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteBucketà la section Référence des AWS SDK pour JavaScript API.

`DeleteBucketPolicy`

L'exemple de code suivant montre comment utiliser`DeleteBucketPolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez la politique du compartiment.

```
import {
  DeleteBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Remove the policy from an Amazon S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteBucketPolicyCommand({
        Bucket: bucketName,
```

```
      }),
    );
    console.log(`Bucket policy deleted from "${bucketName}".`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting policy from ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting policy from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteBucketPolicyà la section Référence des AWS SDK pour JavaScript API.

`DeleteBucketWebsite`

L'exemple de code suivant montre comment utiliser`DeleteBucketWebsite`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez la configuration du site Web du compartiment.

```
import {
  DeleteBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Remove the website configuration for a bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteBucketWebsiteCommand({
        Bucket: bucketName,
      }),
    );
    // The response code will be successful for both removed configurations and
    // configurations that did not exist in the first place.
    console.log(
      `The bucket "${bucketName}" is not longer configured as a website, or it never
 was.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while removing website configuration from ${bucketName}. The
 bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while removing website configuration from ${bucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
```

```
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteBucketWebsiteà la section Référence des AWS SDK pour JavaScript API.

`DeleteObject`

L'exemple de code suivant montre comment utiliser`DeleteObject`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez un objet.

```
import {
  DeleteObjectCommand,
  S3Client,
  S3ServiceException,
  waitUntilObjectNotExists,
} from "@aws-sdk/client-s3";

/**
 * Delete one object from an Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteObjectCommand({
        Bucket: bucketName,
        Key: key,
      }),
```

```
    );
    await waitUntilObjectNotExists(
      { client },
      { Bucket: bucketName, Key: key },
    );
    // A successful delete, or a delete for a non-existent object, both return
    // a 204 response code.
    console.log(
      `The object "${key}" from bucket "${bucketName}" was deleted, or it didn't
 exist.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting object from ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteObjectà la section Référence des AWS SDK pour JavaScript API.

`DeleteObjects`

L'exemple de code suivant montre comment utiliser`DeleteObjects`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez plusieurs objets.

```
import {
  DeleteObjectsCommand,
  S3Client,
  S3ServiceException,
  waitUntilObjectNotExists,
} from "@aws-sdk/client-s3";

/**
 * Delete multiple objects from an S3 bucket.
 * @param {{ bucketName: string, keys: string[] }}
 */
export const main = async ({ bucketName, keys }) => {
  const client = new S3Client({});

  try {
    const { Deleted } = await client.send(
      new DeleteObjectsCommand({
        Bucket: bucketName,
        Delete: {
          Objects: keys.map((k) => ({ Key: k })),
        },
      }),
    );
    for (const key in keys) {
      await waitUntilObjectNotExists(
        { client },
        { Bucket: bucketName, Key: key },
      );
    }
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted
 objects:`,
    );
```

```
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting objects from ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting objects from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteObjectsà la section Référence des AWS SDK pour JavaScript API.

`GetBucketAcl`

L'exemple de code suivant montre comment utiliser`GetBucketAcl`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez les autorisations ACL.

```
import {
  GetBucketAclCommand,
```

```
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Retrieves the Access Control List (ACL) for an S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetBucketAclCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`ACL for bucket "${bucketName}":`);
    console.log(JSON.stringify(response, null, 2));
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting ACL for ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting ACL for ${bucketName}. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetBucketAclà la section Référence des AWS SDK pour JavaScript API.

`GetBucketCors`

L'exemple de code suivant montre comment utiliser`GetBucketCors`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez la politique CORS pour le compartiment.

```
import {
  GetBucketCorsCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the Cross-Origin Resource Sharing (CORS) configuration information
 * set for the bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new GetBucketCorsCommand({
    Bucket: bucketName,
  });

  try {
    const { CORSRules } = await client.send(command);
    console.log(JSON.stringify(CORSRules));
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `\n${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders}`,
        `\nAllowedMethods: ${cr.AllowedMethods}`,
        `\nAllowedOrigins: ${cr.AllowedOrigins}`,
        `\nExposeHeaders: ${cr.ExposeHeaders}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
```

```
      );
    });
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting bucket CORS rules for ${bucketName}. The bucket
 doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting bucket CORS rules for ${bucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetBucketCorsà la section Référence des AWS SDK pour JavaScript API.

`GetBucketPolicy`

L'exemple de code suivant montre comment utiliser`GetBucketPolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez la politique de compartiment.

```
import {
  GetBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Logs the policy for a specified bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { Policy } = await client.send(
      new GetBucketPolicyCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`Policy for "${bucketName}":\n${Policy}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting policy from ${bucketName}. The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting policy from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, reportez-vous GetBucketPolicyà la section Référence des AWS SDK pour JavaScript API.

`GetBucketWebsite`

L'exemple de code suivant montre comment utiliser`GetBucketWebsite`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez la configuration du site web.

```
import {
  GetBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the website configuration for a bucket.
 * @param {{ bucketName }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetBucketWebsiteCommand({
        Bucket: bucketName,
      }),
    );
    console.log(
      `Your bucket is set up to host a website with the following configuration:\n
${JSON.stringify(response, null, 2)}`,
    );
  } catch (caught) {
    if (
```

```
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchWebsiteConfiguration"
    ) {
      console.error(
        `Error from S3 while getting website configuration for ${bucketName}. The
 bucket isn't configured as a website.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting website configuration for ${bucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous GetBucketWebsiteà la section Référence des AWS SDK pour JavaScript API.

`GetObject`

L'exemple de code suivant montre comment utiliser`GetObject`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Téléchargez l'objet.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";
```

```
/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

Téléchargez l'objet à condition qu'il ETag corresponde à celui fourni.

```
import {
  GetObjectCommand,
  NoSuchKey,
```

```
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
```

```
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Téléchargez l'objet à condition ETag qu'il ne corresponde pas à celui fourni.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";
```

```
/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfNoneMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
```

```
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Téléchargez l'objet à condition qu'il ait été créé ou modifié dans un laps de temps donné.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
```

```
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const date = new Date();
  date.setDate(date.getDate() - 1);
  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfModifiedSince: date,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
```

```
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Téléchargez l'objet à condition qu'il n'ait pas été créé ou modifié dans un laps de temps donné.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const date = new Date();
  date.setDate(date.getDate() - 1);
  try {
    const response = await client.send(
      new GetObjectCommand({
```

```
        Bucket: bucketName,
        Key: key,
        IfUnmodifiedSince: date,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
```

```
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetObjectà la section Référence des AWS SDK pour JavaScript API.

`GetObjectLegalHold`

L'exemple de code suivant montre comment utiliser`GetObjectLegalHold`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  GetObjectLegalHoldCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get an object's current legal hold status.
 * @param {{ bucketName: string, key: string }}
```

```
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectLegalHoldCommand({
        Bucket: bucketName,
        Key: key,
        // Optionally, you can provide additional parameters
        // ExpectedBucketOwner: "<account ID that is expected to own the bucket>",
        // VersionId: "<the specific version id of the object to check>",
      }),
    );
    console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting legal hold status for ${key} in ${bucketName}.
 The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting legal hold status for ${key} in ${bucketName}
 from ${bucketName}.  ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
```

```
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus de détails sur l'API, reportez-vous GetObjectLegalHoldà la section Référence des AWS SDK pour JavaScript API.

`GetObjectLockConfiguration`

L'exemple de code suivant montre comment utiliser`GetObjectLockConfiguration`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  GetObjectLockConfigurationCommand,
```

```
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Gets the Object Lock configuration for a bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { ObjectLockConfiguration } = await client.send(
      new GetObjectLockConfigurationCommand({
        Bucket: bucketName,
        // Optionally, you can provide additional parameters
        // ExpectedBucketOwner: "<account ID that is expected to own the bucket>",
      }),
    );
    console.log(
      `Object Lock Configuration:\n${JSON.stringify(ObjectLockConfiguration)}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting object lock configuration for ${bucketName}.
 The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object lock configuration for ${bucketName}.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
```

```
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus de détails sur l'API, reportez-vous GetObjectLockConfigurationà la section Référence des AWS SDK pour JavaScript API.

`GetObjectRetention`

L'exemple de code suivant montre comment utiliser`GetObjectRetention`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  GetObjectRetentionCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the "RetainUntilDate" for an object in an S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const { Retention } = await client.send(
      new GetObjectRetentionCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    console.log(
      `${key} in ${bucketName} will be retained until ${Retention.RetainUntilDate}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchObjectLockConfiguration"
    ) {
      console.warn(
        `The object "${key}" in the bucket "${bucketName}" does not have an
 ObjectLock configuration.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object retention settings for "${bucketName}".
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

```
// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus de détails sur l'API, reportez-vous GetObjectRetentionà la section Référence des AWS SDK pour JavaScript API.

`ListBuckets`

L'exemple de code suivant montre comment utiliser`ListBuckets`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Listez les compartiments.

```
import {
  paginateListBuckets,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * List the Amazon S3 buckets in your account.
 */
export const main = async () => {
  const client = new S3Client({});
  /** @type {?import('@aws-sdk/client-s3').Owner} */
  let Owner = null;

  /** @type {import('@aws-sdk/client-s3').Bucket[]} */
  const Buckets = [];

  try {
    const paginator = paginateListBuckets({ client }, {});

    for await (const page of paginator) {
      if (!Owner) {
        Owner = page.Owner;
      }

      Buckets.push(...page.Buckets);
    }

    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`,
    );
```

```
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (caught) {
    if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while listing buckets.  ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListBucketsà la section Référence des AWS SDK pour JavaScript API.

`ListObjectsV2`

L'exemple de code suivant montre comment utiliser`ListObjectsV2`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez tous les objets dans votre compartiment. S'il existe plusieurs objets, IsTruncated ils NextContinuationToken seront utilisés pour parcourir la liste complète.

```
import {
  S3Client,
  S3ServiceException,
  // This command supersedes the ListObjectsCommand and is the recommended way to
 list objects.
  paginateListObjectsV2,
} from "@aws-sdk/client-s3";

/**
```

```
 * Log all of the object keys in a bucket.
 * @param {{ bucketName: string, pageSize: string }}
 */
export const main = async ({ bucketName, pageSize }) => {
  const client = new S3Client({});
  /** @type {string[][]} */
  const objects = [];
  try {
    const paginator = paginateListObjectsV2(
      { client, /* Max items per page */ pageSize: Number.parseInt(pageSize) },
      { Bucket: bucketName },
    );

    for await (const page of paginator) {
      objects.push(page.Contents.map((o) => o.Key));
    }
    objects.forEach((objectList, pageNum) => {
      console.log(
        `Page ${pageNum + 1}\n------\n${objectList.map((o) => `•
 ${o}`).join("\n")}\n`,
      );
    });
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while listing objects for "${bucketName}". The bucket doesn't
 exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while listing objects for "${bucketName}".  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, consultez la section ListObjectsV2 dans le AWS SDK pour JavaScript manuel de référence des API.

`PutBucketAcl`

L'exemple de code suivant montre comment utiliser`PutBucketAcl`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Effectuez une commande PUT pour l'ACL du compartiment.

```
import {
  PutBucketAclCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Grant read access to a user using their canonical AWS account ID.
 *
 * Most Amazon S3 use cases don't require the use of access control lists (ACLs).
 * We recommend that you disable ACLs, except in unusual circumstances where
 * you need to control access for each object individually. Consider a policy
 instead.
 * For more information see https://docs.aws.amazon.com/AmazonS3/latest/userguide/
bucket-policies.html.
 * @param {{ bucketName: string, granteeCanonicalUserId: string,
 ownerCanonicalUserId }}
 */
export const main = async ({
  bucketName,
  granteeCanonicalUserId,
  ownerCanonicalUserId,
}) => {
  const client = new S3Client({});
  const command = new PutBucketAclCommand({
```

```
    Bucket: bucketName,
    AccessControlPolicy: {
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
 AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
            ID: granteeCanonicalUserId,
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "READ",
        },
      ],
      Owner: {
        ID: ownerCanonicalUserId,
      },
    },
  });

  try {
    await client.send(command);
    console.log(`Granted READ access to ${bucketName}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while setting ACL for bucket ${bucketName}. The bucket
 doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting ACL for bucket ${bucketName}. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
```

```
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutBucketAclà la section Référence des AWS SDK pour JavaScript API.

`PutBucketCors`

L'exemple de code suivant montre comment utiliser`PutBucketCors`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Ajoutez une règle CORS.

```
import {
  PutBucketCorsCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Allows cross-origin requests to an S3 bucket by setting the CORS configuration.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new PutBucketCorsCommand({
        Bucket: bucketName,
        CORSConfiguration: {
```

```
          CORSRules: [
            {
              // Allow all headers to be sent to this bucket.
              AllowedHeaders: ["*"],
              // Allow only GET and PUT methods to be sent to this bucket.
              AllowedMethods: ["GET", "PUT"],
              // Allow only requests from the specified origin.
              AllowedOrigins: ["https://www.example.com"],
              // Allow the entity tag (ETag) header to be returned in the response.
 The ETag header
              // The entity tag represents a specific version of the object. The
 ETag reflects
              // changes only to the contents of an object, not its metadata.
              ExposeHeaders: ["ETag"],
              // How long the requesting browser should cache the preflight
 response. After
              // this time, the preflight request will have to be made again.
              MaxAgeSeconds: 3600,
            },
          ],
        },
      }),
    );
    console.log(`Successfully set CORS rules for bucket: ${bucketName}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while setting CORS rules for ${bucketName}. The bucket
 doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting CORS rules for ${bucketName}. ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutBucketCorsà la section Référence des AWS SDK pour JavaScript API.

`PutBucketPolicy`

L'exemple de code suivant montre comment utiliser`PutBucketPolicy`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Ajoutez la politique.

```
import {
  PutBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Grant an IAM role GetObject access to all of the objects
 * in the provided bucket.
 * @param {{ bucketName: string, iamRoleArn: string }}
 */
export const main = async ({ bucketName, iamRoleArn }) => {
  const client = new S3Client({});
  const command = new PutBucketPolicyCommand({
    // This is a resource-based policy. For more information on resource-based
 policies,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/
access_policies.html#policies_resource-based.
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
```

```
        {
          Effect: "Allow",
          Principal: {
            AWS: iamRoleArn,
          },
          Action: "s3:GetObject",
          Resource: `arn:aws:s3:::${bucketName}/*`,
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: bucketName,
  });

  try {
    await client.send(command);
    console.log(
      `GetObject access to the bucket "${bucketName}" was granted to the provided
 IAM role.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "MalformedPolicy"
    ) {
      console.error(
        `Error from S3 while setting the bucket policy for the bucket
 "${bucketName}". The policy was malformed.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting the bucket policy for the bucket
 "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, reportez-vous PutBucketPolicyà la section Référence des AWS SDK pour JavaScript API.

`PutBucketWebsite`

L'exemple de code suivant montre comment utiliser`PutBucketWebsite`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Définissez la configuration du site web.

```
import {
  PutBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Configure an Amazon S3 bucket to serve a static website.
 * Website access must also be granted separately. For more information
 * on setting the permissions for website access, see
 * https://docs.aws.amazon.com/AmazonS3/latest/userguide/
WebsiteAccessPermissionsReqd.html.
 *
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new PutBucketWebsiteCommand({
    Bucket: bucketName,
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
```

```
        // A suffix that is appended to a request when the request is
        // for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
    await client.send(command);
    console.log(
      `The bucket "${bucketName}" has been configured as a static website.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while configuring the bucket "${bucketName}" as a static
 website. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while configuring the bucket "${bucketName}" as a static
 website. ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutBucketWebsiteà la section Référence des AWS SDK pour JavaScript API.

`PutObject`

L'exemple de code suivant montre comment utiliser`PutObject`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Chargez l'objet.

```
import { readFile } from "node:fs/promises";

import {
  PutObjectCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Upload a file to an S3 bucket.
 * @param {{ bucketName: string, key: string, filePath: string }}
 */
export const main = async ({ bucketName, key, filePath }) => {
  const client = new S3Client({});
  const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: key,
    Body: await readFile(filePath),
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "EntityTooLarge"
    ) {
      console.error(
        `Error from S3 while uploading object to ${bucketName}. \
The object was too large. To upload objects larger than 5GB, use the S3 console
 (160GB max) \
or the multipart upload API (5TB max).`,
```

```
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while uploading object to ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

Téléchargez l'objet à condition qu'il ETag corresponde à celui fourni.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
```

```
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
 key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}.  ${caught.name}:
 ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
```

```
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutObjectà la section Référence des AWS SDK pour JavaScript API.

`PutObjectLegalHold`

L'exemple de code suivant montre comment utiliser`PutObjectLegalHold`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  PutObjectLegalHoldCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Apply a legal hold configuration to the specified object.
 * @param {{ bucketName: string, objectKey: string, legalHoldStatus: "ON" | "OFF" }}
 */
export const main = async ({ bucketName, objectKey, legalHoldStatus }) => {
  if (!["OFF", "ON"].includes(legalHoldStatus.toUpperCase())) {
    throw new Error(
      "Invalid parameter. legalHoldStatus must be 'ON' or 'OFF'.",
    );
  }
```

```
  const client = new S3Client({});
  const command = new PutObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    LegalHold: {
      // Set the status to 'ON' to place a legal hold on the object.
      // Set the status to 'OFF' to remove the legal hold.
      Status: legalHoldStatus,
    },
  });

  try {
    await client.send(command);
    console.log(
      `Legal hold status set to "${legalHoldStatus}" for "${objectKey}" in
 "${bucketName}"`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying legal hold status for "${objectKey}" in
 "${bucketName}". The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying legal hold status for "${objectKey}" in
 "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";
```

```
const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    objectKey: {
      type: "string",
      required: true,
    },
    legalHoldStatus: {
      type: "string",
      default: "ON",
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus de détails sur l'API, reportez-vous PutObjectLegalHoldà la section Référence des AWS SDK pour JavaScript API.

`PutObjectLockConfiguration`

L'exemple de code suivant montre comment utiliser`PutObjectLockConfiguration`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Définissez la configuration du verrouillage d'objet d'un compartiment.

```
import {
  PutObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Enable S3 Object Lock for an Amazon S3 bucket.
 * After you enable Object Lock on a bucket, you can't
 * disable Object Lock or suspend versioning for that bucket.
 * @param {{ bucketName: string, enabled: boolean }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
  });

  try {
    await client.send(command);
    console.log(`Object Lock for "${bucketName}" enabled.`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying the object lock configuration for the bucket
 "${bucketName}". The bucket doesn't exist.`,
```

```
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying the object lock configuration for the bucket
 "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

Définissez la période de rétention par défaut d’un compartiment.

```
import {
  PutObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Change the default retention settings for an object in an Amazon S3 bucket.
 * @param {{ bucketName: string, retentionDays: string }}
 */
export const main = async ({ bucketName, retentionDays }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: bucketName,
        // The Object Lock configuration that you want to apply to the specified
 bucket.
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
          Rule: {
            // The default Object Lock retention mode and period that you want to
 apply
            // to new objects placed in the specified bucket. Bucket settings
 require
            // both a mode and a period. The period can be either Days or Years but
            // you must select one.
            DefaultRetention: {
              // In governance mode, users can't overwrite or delete an object
 version
              // or alter its lock settings unless they have special permissions.
 With
              // governance mode, you protect objects against being deleted by most
 users,
              // but you can still grant some users permission to alter the
 retention settings
              // or delete the objects if necessary.
              Mode: "GOVERNANCE",
              Days: Number.parseInt(retentionDays),
            },
          },
```

```
        },
      }),
    );
    console.log(
      `Set default retention mode to "GOVERNANCE" with a retention period of
 ${retentionDays} day(s).`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while setting the default object retention for a bucket. The
 bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting the default object retention for a bucket.
 ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    retentionDays: {
      type: "string",
      required: true,
    },
```

```
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus de détails sur l'API, reportez-vous PutObjectLockConfigurationà la section Référence des AWS SDK pour JavaScript API.

`PutObjectRetention`

L'exemple de code suivant montre comment utiliser`PutObjectRetention`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  PutObjectRetentionCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Place a 24-hour retention period on an object in an Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
```

```
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: key,
    BypassGovernanceRetention: false,
    Retention: {
      // In governance mode, users can't overwrite or delete an object version
      // or alter its lock settings unless they have special permissions. With
      // governance mode, you protect objects against being deleted by most users,
      // but you can still grant some users permission to alter the retention
 settings
      // or delete the objects if necessary.
      Mode: "GOVERNANCE",
      RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
  });

  try {
    await client.send(command);
    console.log("Object Retention settings updated.");
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying the governance mode and retention period on
 an object. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying the governance mode and retention period on
 an object. ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
```

```
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Pour plus de détails sur l'API, reportez-vous PutObjectRetentionà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer une URL présignée

L'exemple de code suivant montre comment créer une URL présignée pour Amazon S3 et charger un objet.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une URL présignée pour charger un objet dans un compartiment.

```
import https from "node:https";

import { XMLParser } from "fast-xml-parser";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};
```

```
/**
 * Make a PUT request to the provided URL.
 *
 * @param {string} url
 * @param {string} data
 */
const put = (url, data) => {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          const parser = new XMLParser();
          if (res.statusCode >= 200 && res.statusCode <= 299) {
            resolve(parser.parse(responseBody, true));
          } else {
            reject(parser.parse(responseBody, true));
          }
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
};

/**
 * Create two presigned urls for uploading an object to an S3 bucket.
 * The first presigned URL is created with credentials from the shared INI file
 * in the current environment. The second presigned URL is created using an
 * existing S3Client instance that has already been provided with credentials.
 * @param {{ bucketName: string, key: string, region: string }}
 */
export const main = async ({ bucketName, key, region }) => {
  try {
```

```
    const noClientUrl = await createPresignedUrlWithoutClient({
      bucket: bucketName,
      key,
      region,
    });

    const clientUrl = await createPresignedUrlWithClient({
      bucket: bucketName,
      region,
      key,
    });

    // After you get the presigned URL, you can provide your own file
    // data. Refer to put() above.
    console.log("Calling PUT using presigned URL without client");
    await put(noClientUrl, "Hello World");

    console.log("Calling PUT using presigned URL with client");
    await put(clientUrl, "Hello World");

    console.log("\nDone. Check your S3 console.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "CredentialsProviderError") {
      console.error(
        `There was an error getting your credentials. Are your local credentials
 configured?\n${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

Créez une URL présignée pour télécharger un objet à partir d'un compartiment.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
```

```
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(new HttpRequest(url));
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new GetObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

/**
 * Create two presigned urls for downloading an object from an S3 bucket.
 * The first presigned URL is created with credentials from the shared INI file
 * in the current environment. The second presigned URL is created using an
 * existing S3Client instance that has already been provided with credentials.
 * @param {{ bucketName: string, key: string, region: string }}
 */
export const main = async ({ bucketName, key, region }) => {
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      bucket: bucketName,
      region,
      key,
    });

    const clientUrl = await createPresignedUrlWithClient({
      bucket: bucketName,
      region,
      key,
    });

    console.log("Presigned URL without client");
```

```
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "CredentialsProviderError") {
      console.error(
        `There was an error getting your credentials. Are your local credentials
 configured?\n${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour JavaScript (v3)

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

Créer une page web qui répertorie les objets Amazon S3

L'exemple de code suivant montre comment répertorier des objets Amazon S3 dans une page web.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Le code suivant est le composant React pertinent qui appelle le AWS SDK. Une version exécutable de l'application contenant ce composant se trouve sur le lien précédent GitHub .

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  type ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";

function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
 role to the pool,
      // and grant the role access to the 's3:GetObject' action.
```

```
      //
      // You'll also need to configure the CORS settings on the bucket to allow
 traffic from
      // this example site. Here's an example configuration that allows all origins.
 Don't
      // do this in production.
      //[
      //  {
      //    "AllowedHeaders": ["*"],
      //    "AllowedMethods": ["GET"],
      //    "AllowedOrigins": ["*"],
      //    "ExposeHeaders": [],
      //  },
      //]
      //
      credentials: fromCognitoIdentityPool({
        clientConfig: { region: "us-east-1" },
        identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
      }),
    });
    const command = new ListObjectsCommand({ Bucket: "bucket-name" });
    client.send(command).then(({ Contents }) => setObjects(Contents || []));
  }, []);

  return (
    <div className="App">
      {objects.map((o) => (
        <div key={o.ETag}>{o.Key}</div>
      ))}
    </div>
  );
}

export default App;
```

- Pour plus de détails sur l'API, reportez-vous ListObjectsà la section Référence des AWS SDK pour JavaScript API.

Créer une application Amazon Textract Explorer

L'exemple de code suivant montre comment explorer la sortie Amazon Textract via une application interactive.

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK pour JavaScript pour créer une application React qui utilise Amazon Textract pour extraire des données d'une image de document et les afficher sur une page Web interactive. Cet exemple s'exécute dans un navigateur Web et nécessite une identité Amazon Cognito authentifiée pour les informations d'identification. Il utilise Amazon Simple Storage Service (Amazon S3) pour le stockage et, pour les notifications, il interroge une file d'attente Amazon Simple Queue Service (Amazon SQS) abonnée à une rubrique Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Suppression de tous les objets dans un compartiment

L'exemple de code suivant montre comment supprimer tous les objets d'un compartiment Amazon S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez tous les objets d'un compartiment Amazon S3 donné.

```
import {
  DeleteObjectsCommand,
  paginateListObjectsV2,
```

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

/**
 *
 * @param {{ bucketName: string }} config
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  try {
    console.log(`Deleting all objects in bucket: ${bucketName}`);

    const paginator = paginateListObjectsV2(
      { client },
      {
        Bucket: bucketName,
      },
    );

    const objectKeys = [];
    for await (const { Contents } of paginator) {
      objectKeys.push(...Contents.map((obj) => ({ Key: obj.Key })));
    }

    const deleteCommand = new DeleteObjectsCommand({
      Bucket: bucketName,
      Delete: { Objects: objectKeys },
    });

    await client.send(deleteCommand);

    console.log(`All objects deleted from bucket: ${bucketName}`);
  } catch (caught) {
    if (caught instanceof Error) {
      console.error(
        `Failed to empty ${bucketName}. ${caught.name}: ${caught.message}`,
      );
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    bucketName: {
      type: "string",
    },
  };

  const { values } = parseArgs({ options });
  main(values);
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - DeleteObjects
  - ListObjectsV2

Détecter des objets dans des images

L'exemple de code suivant montre comment créer une application qui utilise Amazon Rekognition afin de détecter des objets par catégorie dans des images.

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Rekognition AWS SDK pour JavaScript pour créer une application qui utilise Amazon Rekognition pour identifier les objets par catégorie dans des images situées dans un compartiment Amazon Simple Storage Service (Amazon S3). L'application envoie à l'administrateur une notification par e-mail contenant les résultats à l'aide d'Amazon Simple Email Service (Amazon SES).

Découvrez comment :

- Créer un utilisateur non authentifié à l'aide d'Amazon Cognito.
- Analyser les images à la recherche d'objets à l'aide d'Amazon Rekognition.
- Vérifier une adresse e-mail pour Amazon SES.
- Envoyer une notification par e-mail à l'aide d'Amazon SES.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Rekognition
- Amazon S3
- Amazon SES

Verrouillage d'objets Amazon S3

L'exemple de code suivant montre comment travailler avec les fonctionnalités de verrouillage d'objets S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Point d'entrée pour le scénario (index.js). Cela orchestre toutes les étapes. Consultez GitHub les détails de mise en œuvre de Scenario ScenarioInput, ScenarioOutput, et ScenarioAction.

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "./welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  confirmSetLegalHoldFileEnabled,
  confirmSetLegalHoldFileRetention,
  confirmSetRetentionPeriodFileEnabled,
  confirmSetRetentionPeriodFileRetention,
  confirmUpdateLockPolicy,
  confirmUpdateRetention,
  createBuckets,
  createBucketsAction,
```

```
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
  setLegalHoldFileEnabledAction,
  setLegalHoldFileRetentionAction,
  setRetentionPeriodFileEnabledAction,
  setRetentionPeriodFileRetentionAction,
  updateLockPolicy,
  updateLockPolicyAction,
  updateRetention,
  updateRetentionAction,
} from "./setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
    deploy: new scenarios.Scenario(
      "S3 Object Locking - Deploy",
      [
        welcome(scenarios),
        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        getBucketPrefix(scenarios),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
        createBucketsAction(scenarios, client),
        updateRetention(scenarios),
        confirmUpdateRetention(scenarios),
        exitOnFalse(scenarios, "confirmUpdateRetention"),
        updateRetentionAction(scenarios, client),
        populateBuckets(scenarios),
        confirmPopulateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        updateLockPolicy(scenarios),
        confirmUpdateLockPolicy(scenarios),
        exitOnFalse(scenarios, "confirmUpdateLockPolicy"),
        updateLockPolicyAction(scenarios, client),
```

```
        confirmSetLegalHoldFileEnabled(scenarios),
        setLegalHoldFileEnabledAction(scenarios, client),
        confirmSetRetentionPeriodFileEnabled(scenarios),
        setRetentionPeriodFileEnabledAction(scenarios, client),
        confirmSetLegalHoldFileRetention(scenarios),
        setLegalHoldFileRetentionAction(scenarios, client),
        confirmSetRetentionPeriodFileRetention(scenarios),
        setRetentionPeriodFileRetentionAction(scenarios, client),
        saveState,
      ],
      initialState,
    ),
    demo: new scenarios.Scenario(
      "S3 Object Locking - Demo",
      [loadState, replAction(scenarios, client)],
      initialState,
    ),
    clean: new scenarios.Scenario(
      "S3 Object Locking - Destroy",
      [
        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
      ],
      initialState,
    ),
  };
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios, {
    name: "Amazon S3 object locking workflow",
    description:
      "Work with Amazon Simple Storage Service (Amazon S3) object locking
 features.",
    synopsis:
```

```
      "node index.js --scenario <deploy | demo | clean> [-h|--help] [-y|--yes] [-
v|--verbose]",
  });
}
```

Envoyez des messages de bienvenue à la console (welcome.steps.js).

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    "Welcome to the Amazon Simple Storage Service (S3) Object Locking Feature
 Scenario. For this workflow, we will use the AWS SDK for JavaScript to create
 several S3 buckets and files to demonstrate working with S3 locking features.",
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };
```

Déployez des compartiments, des objets et des paramètres de fichier (setup.steps.js).

```
import {
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
```

```
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,
  GetBucketVersioningCommand,
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  S3ServiceException,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const getBucketPrefix = (scenarios) =>
  new scenarios.ScenarioInput(
    "bucketPrefix",
    "Provide a prefix that will be used for bucket creation.",
    { type: "input", default: "amzn-s3-demo-bucket" },
  );

/**
 * @param {Scenarios} scenarios
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    (state) => `The following buckets will be created:
         ${state.bucketPrefix}-no-lock with object lock False.
         ${state.bucketPrefix}-lock-enabled with object lock True.
```

```
        ${state.bucketPrefix}-retention-after-creation with object lock False.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const noLockBucketName = `${state.bucketPrefix}-no-lock`;
    const lockEnabledBucketName = `${state.bucketPrefix}-lock-enabled`;
    const retentionBucketName = `${state.bucketPrefix}-retention-after-creation`;

    try {
      await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
      await waitUntilBucketExists({ client }, { Bucket: noLockBucketName });
      await client.send(
        new CreateBucketCommand({
          Bucket: lockEnabledBucketName,
          ObjectLockEnabledForBucket: true,
        }),
      );
      await waitUntilBucketExists(
        { client },
        { Bucket: lockEnabledBucketName },
      );
      await client.send(
        new CreateBucketCommand({ Bucket: retentionBucketName }),
      );
      await waitUntilBucketExists({ client }, { Bucket: retentionBucketName });

      state.noLockBucketName = noLockBucketName;
      state.lockEnabledBucketName = lockEnabledBucketName;
      state.retentionBucketName = retentionBucketName;
    } catch (caught) {
```

```
      if (
        caught instanceof BucketAlreadyExists ||
        caught instanceof BucketAlreadyOwnedByYou
      ) {
        console.error(`${caught.name}: ${caught.message}`);
        state.earlyExit = true;
      } else {
        throw caught;
      }
    }
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    (state) => `The following test files will be created:
         file0.txt in ${state.bucketPrefix}-no-lock.
         file1.txt in ${state.bucketPrefix}-no-lock.
         file0.txt in ${state.bucketPrefix}-lock-enabled.
         file1.txt in ${state.bucketPrefix}-lock-enabled.
         file0.txt in ${state.bucketPrefix}-retention-after-creation.
         file1.txt in ${state.bucketPrefix}-retention-after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
```

```
    try {
      await client.send(
        new PutObjectCommand({
          Bucket: state.noLockBucketName,
          Key: "file0.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
      await client.send(
        new PutObjectCommand({
          Bucket: state.noLockBucketName,
          Key: "file1.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
      await client.send(
        new PutObjectCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file0.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
      await client.send(
        new PutObjectCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file1.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
      await client.send(
        new PutObjectCommand({
          Bucket: state.retentionBucketName,
          Key: "file0.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
      await client.send(
        new PutObjectCommand({
          Bucket: state.retentionBucketName,
```

```
          Key: "file1.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
    } catch (caught) {
      if (caught instanceof S3ServiceException) {
        console.error(
          `Error from S3 while uploading object.  ${caught.name}:
 ${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  });

/**
 * @param {Scenarios} scenarios
 */
const updateRetention = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateRetention",
    (state) => `A bucket can be configured to use object locking with a default
 retention period.
A default retention period will be configured for ${state.bucketPrefix}-retention-
after-creation.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateRetention",
    "Configure default retention period?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
```

```
const updateRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateRetentionAction", async (state) => {
    await client.send(
      new PutBucketVersioningCommand({
        Bucket: state.retentionBucketName,
        VersioningConfiguration: {
          MFADelete: MFADeleteStatus.Disabled,
          Status: BucketVersioningStatus.Enabled,
        },
      }),
    );

    const getBucketVersioning = new GetBucketVersioningCommand({
      Bucket: state.retentionBucketName,
    });

    await retry({ intervalInMs: 500, maxRetries: 10 }, async () => {
      const { Status } = await client.send(getBucketVersioning);
      if (Status !== "Enabled") {
        throw new Error("Bucket versioning is not enabled.");
      }
    });

    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.retentionBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
          Rule: {
            DefaultRetention: {
              Mode: "GOVERNANCE",
              Years: 1,
            },
          },
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateLockPolicy = (scenarios) =>
  new scenarios.ScenarioOutput(
```

```
    "updateLockPolicy",
    (state) => `Object lock policies can also be added to existing buckets.
An object lock policy will be added to ${state.bucketPrefix}-lock-enabled.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateLockPolicy = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateLockPolicy",
    "Add object lock policy?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateLockPolicyAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateLockPolicyAction", async (state) => {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.lockEnabledBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileEnabled",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.lockEnabledBucketName}?`,
    {
      type: "confirm",
```

```
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileEnabledAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.lockEnabledBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetRetentionPeriodFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileEnabled",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.lockEnabledBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );
```

```
/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileEnabledAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
        }),
      );
      console.log(
        `Set retention for file1.txt in ${state.lockEnabledBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
 ${state.retentionBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
```

```
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileRetentionAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.retentionBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.retentionBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileRetention },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmSetRetentionPeriodFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileRetention",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
 ${state.retentionBucketName}?
Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
 to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileRetentionAction = (scenarios, client) =>
```

```
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileRetentionAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.retentionBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
          BypassGovernanceRetention: true,
        }),
      );
      console.log(
        `Set retention for file1.txt in ${state.retentionBucketName} until
 ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileRetention },
  );

export {
  getBucketPrefix,
  createBuckets,
  confirmCreateBuckets,
  createBucketsAction,
  populateBuckets,
  confirmPopulateBuckets,
  populateBucketsAction,
  updateRetention,
  confirmUpdateRetention,
  updateRetentionAction,
  updateLockPolicy,
  confirmUpdateLockPolicy,
  updateLockPolicyAction,
  confirmSetLegalHoldFileEnabled,
  setLegalHoldFileEnabledAction,
  confirmSetRetentionPeriodFileEnabled,
  setRetentionPeriodFileEnabledAction,
  confirmSetLegalHoldFileRetention,
  setLegalHoldFileRetentionAction,
```

```
  confirmSetRetentionPeriodFileRetention,
  setRetentionPeriodFileRetentionAction,
};
```

Affichez et supprimez des fichiers dans les compartiments (repl.steps.js).

```
import {
  ChecksumAlgorithm,
  DeleteObjectCommand,
  GetObjectLegalHoldCommand,
  GetObjectLockConfigurationCommand,
  GetObjectRetentionCommand,
  ListObjectVersionsCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  DELETE_FILE: 2,
  DELETE_FILE_WITH_RETENTION: 3,
  OVERWRITE_FILE: 4,
  VIEW_RETENTION_SETTINGS: 5,
  VIEW_LEGAL_HOLD_SETTINGS: 6,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new scenarios.ScenarioInput(
    "replChoice",
    "Explore the S3 locking features by selecting one of the following choices",
    {
```

```
      type: "select",
      choices: [
        { name: "List all files in buckets", value: choices.LIST_ALL_FILES },
        { name: "Attempt to delete a file.", value: choices.DELETE_FILE },
        {
          name: "Attempt to delete a file with retention period bypass.",
          value: choices.DELETE_FILE_WITH_RETENTION,
        },
        { name: "Attempt to overwrite a file.", value: choices.OVERWRITE_FILE },
        {
          name: "View the object and bucket retention settings for a file.",
          value: choices.VIEW_RETENTION_SETTINGS,
        },
        {
          name: "View the legal hold settings for a file.",
          value: choices.VIEW_LEGAL_HOLD_SETTINGS,
        },
        { name: "Finish the workflow.", value: choices.EXIT },
      ],
    },
  );

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;
      files.push({ bucket, key: Key, version: VersionId });
    }
  }

  return files;
};

/**
 * @param {Scenarios} scenarios
```

```
 * @param {S3Client} client
 */
const replAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.noLockBucketName,
        state.lockEnabledBucketName,
        state.retentionBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file:",
        {
          type: "select",
          choices: files.map((file, index) => ({
            name: `${index + 1}: ${file.bucket}: ${file.key} (version: ${
              file.version
            })`,
            value: index,
          })),
        },
      );

      const { replChoice } = state;

      switch (replChoice) {
        case choices.LIST_ALL_FILES: {
          const files = await getAllFiles(client, [
            state.noLockBucketName,
            state.lockEnabledBucketName,
            state.retentionBucketName,
          ]);
          state.replOutput = files
            .map(
              (file) =>
                `${file.bucket}: ${file.key} (version: ${file.version})`,
            )
            .join("\n");
          break;
        }
        case choices.DELETE_FILE: {
```

```
          /** @type {number} */
          const fileToDelete = await fileInput.handle(state);
          const selectedFile = files[fileToDelete];
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.DELETE_FILE_WITH_RETENTION: {
          /** @type {number} */
          const fileToDelete = await fileInput.handle(state);
          const selectedFile = files[fileToDelete];
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
                BypassGovernanceRetention: true,
              }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.OVERWRITE_FILE: {
          /** @type {number} */
          const fileToOverwrite = await fileInput.handle(state);
          const selectedFile = files[fileToOverwrite];
```

```
          try {
            await client.send(
              new PutObjectCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                Body: "New content",
                ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
              }),
            );
            state.replOutput = `Overwrote ${selectedFile.key} in
 ${selectedFile.bucket}.`;
          } catch (err) {
            state.replOutput = `Unable to overwrite object ${selectedFile.key} in
 bucket ${selectedFile.bucket}: ${err.message}`;
          }
          break;
        }
        case choices.VIEW_RETENTION_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
            const retention = await client.send(
              new GetObjectRetentionCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            const bucketConfig = await client.send(
              new GetObjectLockConfigurationCommand({
                Bucket: selectedFile.bucket,
              }),
            );
            state.replOutput = `Object retention for ${selectedFile.key}
 in ${selectedFile.bucket}: ${retention.Retention?.Mode} until
 ${retention.Retention?.RetainUntilDate?.toISOString()}.
Bucket object lock config for ${selectedFile.bucket} in ${selectedFile.bucket}:
Enabled: ${bucketConfig.ObjectLockConfiguration?.ObjectLockEnabled}
Rule:
 ${JSON.stringify(bucketConfig.ObjectLockConfiguration?.Rule?.DefaultRetention)}`;
          } catch (err) {
            state.replOutput = `Unable to fetch object lock retention:
 '${err.message}'`;
```

```
          }
          break;
        }
        case choices.VIEW_LEGAL_HOLD_SETTINGS: {
          /** @type {number} */
          const fileToView = await fileInput.handle(state);
          const selectedFile = files[fileToView];
          try {
            const legalHold = await client.send(
              new GetObjectLegalHoldCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
              }),
            );
            state.replOutput = `Object legal hold for ${selectedFile.key} in
 ${selectedFile.bucket}: Status: ${legalHold.LegalHold?.Status}`;
          } catch (err) {
            state.replOutput = `Unable to fetch legal hold: '${err.message}'`;
          }
          break;
        }
        default:
          throw new Error(`Invalid replChoice: ${replChoice}`);
      }
    },
    {
      whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new scenarios.ScenarioOutput(
          "REPL output",
          (state) => state.replOutput,
          { preformatted: true },
        ),
      },
    },
  );

export { replInput, replAction, choices };
```

Détruisez toutes les ressources créées (clean.steps.js).

```
import {
  DeleteObjectCommand,
  DeleteBucketCommand,
  ListObjectVersionsCommand,
  GetObjectLegalHoldCommand,
  GetObjectRetentionCommand,
  PutObjectLegalHoldCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { noLockBucketName, lockEnabledBucketName, retentionBucketName } =
      state;

    const buckets = [
      noLockBucketName,
      lockEnabledBucketName,
      retentionBucketName,
    ];

    for (const bucket of buckets) {
      /** @type {import("@aws-sdk/client-s3").ListObjectVersionsCommandOutput} */
      let objectsResponse;
```

```
  try {
    objectsResponse = await client.send(
      new ListObjectVersionsCommand({
        Bucket: bucket,
      }),
    );
  } catch (e) {
    if (e instanceof Error && e.name === "NoSuchBucket") {
      console.log("Object's bucket has already been deleted.");
      continue;
    }
    throw e;
  }

  for (const version of objectsResponse.Versions || []) {
    const { Key, VersionId } = version;

    try {
      const legalHold = await client.send(
        new GetObjectLegalHoldCommand({
          Bucket: bucket,
          Key,
          VersionId,
        }),
      );

      if (legalHold.LegalHold?.Status === "ON") {
        await client.send(
          new PutObjectLegalHoldCommand({
            Bucket: bucket,
            Key,
            VersionId,
            LegalHold: {
              Status: "OFF",
            },
          }),
        );
      }
    } catch (err) {
      console.log(
        `Unable to fetch legal hold for ${Key} in ${bucket}: '${err.message}'`,
      );
    }
```

```
        try {
          const retention = await client.send(
            new GetObjectRetentionCommand({
              Bucket: bucket,
              Key,
              VersionId,
            }),
          );

          if (retention.Retention?.Mode === "GOVERNANCE") {
            await client.send(
              new DeleteObjectCommand({
                Bucket: bucket,
                Key,
                VersionId,
                BypassGovernanceRetention: true,
              }),
            );
          }
        } catch (err) {
          console.log(
            `Unable to fetch object lock retention for ${Key} in ${bucket}:
 '${err.message}'`,
          );
        }

        await client.send(
          new DeleteObjectCommand({
            Bucket: bucket,
            Key,
            VersionId,
          }),
        );
      }

      await client.send(new DeleteBucketCommand({ Bucket: bucket }));
      console.log(`Delete for ${bucket} complete.`);
    }
  });

export { confirmCleanup, cleanupAction };
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold
  - PutObjectLockConfiguration
  - PutObjectRetention

Réalisation de demandes conditionnelles

L'exemple de code suivant montre comment ajouter des conditions préalables aux demandes Amazon S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Point d'entrée pour le flux de travail (index.js). Cela orchestre toutes les étapes. Consultez GitHub les détails de mise en œuvre de Scenario ScenarioInput, ScenarioOutput, et ScenarioAction.

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "./welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
```

```
  populateBuckets,
  populateBucketsAction,
} from "./setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
    deploy: new scenarios.Scenario(
      "S3 Conditional Requests - Deploy",
      [
        welcome(scenarios),
        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        getBucketPrefix(scenarios),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
        createBucketsAction(scenarios, client),
        populateBuckets(scenarios),
        confirmPopulateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        saveState,
      ],
      initialState,
    ),
    demo: new scenarios.Scenario(
      "S3 Conditional Requests - Demo",
      [loadState, welcome(scenarios), replAction(scenarios, client)],
      initialState,
    ),
    clean: new scenarios.Scenario(
      "S3 Conditional Requests - Destroy",
      [
        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
      ],
```

```
      initialState,
    ),
  };
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios, {
    name: "Amazon S3 object locking workflow",
    description:
      "Work with Amazon Simple Storage Service (Amazon S3) object locking
 features.",
    synopsis:
      "node index.js --scenario <deploy | demo | clean> [-h|--help] [-y|--yes] [-
v|--verbose]",
  });
}
```

Envoyez des messages de bienvenue à la console (welcome.steps.js).

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    "This example demonstrates the use of conditional requests for S3 operations." +
      " You can use conditional requests to add preconditions to S3 read requests to
 return " +
      "or copy an object based on its Entity tag (ETag), or last modified date.You
 can use " +
```

```
      "a conditional write requests to prevent overwrites by ensuring there is no
 existing " +
      "object with the same key.\n" +
      "This example will enable you to perform conditional reads and writes that
 will succeed " +
      "or fail based on your selected options.\n" +
      "Sample buckets and a sample object will be created as part of the example.\n"
 +
      "Some steps require a key name prefix to be defined by the user. Before you
 begin, you can " +
      "optionally edit this prefix in ./object_name.json. If you do so, please
 reload the scenario before you begin.",
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };
```

Déployez des compartiments et des objets (setup.steps.js).

```
import {
  ChecksumAlgorithm,
  CreateBucketCommand,
  PutObjectCommand,
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  S3ServiceException,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */
```

```
/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const getBucketPrefix = (scenarios) =>
  new scenarios.ScenarioInput(
    "bucketPrefix",
    "Provide a prefix that will be used for bucket creation.",
    { type: "input", default: "amzn-s3-demo-bucket" },
  );
/**
 * @param {Scenarios} scenarios
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    (state) => `The following buckets will be created:
         ${state.bucketPrefix}-source-bucket.
         ${state.bucketPrefix}-destination-bucket.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const sourceBucketName = `${state.bucketPrefix}-source-bucket`;
    const destinationBucketName = `${state.bucketPrefix}-destination-bucket`;

    try {
```

```
      await client.send(
        new CreateBucketCommand({
          Bucket: sourceBucketName,
        }),
      );
      await waitUntilBucketExists({ client }, { Bucket: sourceBucketName });
      await client.send(
        new CreateBucketCommand({
          Bucket: destinationBucketName,
        }),
      );
      await waitUntilBucketExists(
        { client },
        { Bucket: destinationBucketName },
      );

      state.sourceBucketName = sourceBucketName;
      state.destinationBucketName = destinationBucketName;
    } catch (caught) {
      if (
        caught instanceof BucketAlreadyExists ||
        caught instanceof BucketAlreadyOwnedByYou
      ) {
        console.error(`${caught.name}: ${caught.message}`);
        state.earlyExit = true;
      } else {
        throw caught;
      }
    }
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    (state) => `The following test files will be created:
         file01.txt in ${state.bucketPrefix}-source-bucket.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
```

```
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
    try {
      await client.send(
        new PutObjectCommand({
          Bucket: state.sourceBucketName,
          Key: "file01.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
    } catch (caught) {
      if (caught instanceof S3ServiceException) {
        console.error(
          `Error from S3 while uploading object.  ${caught.name}:
 ${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  });

export {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
};
```

Obtenez, copiez et placez des objets à l'aide des requêtes conditionnelles S3 (repl.steps.js).

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

import {
  ListObjectVersionsCommand,
  GetObjectCommand,
  CopyObjectCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";
import data from "./object_name.json" assert { type: "json" };
import { readFile } from "node:fs/promises";
import {
  ScenarioInput,
  Scenario,
  ScenarioAction,
  ScenarioOutput,
} from "../../../libs/scenario/index.js";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  CONDITIONAL_READ: 2,
  CONDITIONAL_COPY: 3,
  CONDITIONAL_WRITE: 4,
};

/**
 * @param {Scenarios} scenarios
 */
```

```
const replInput = (scenarios) =>
  new ScenarioInput(
    "replChoice",
    "Explore the S3 conditional request features by selecting one of the following
 choices",
    {
      type: "select",
      choices: [
        { name: "Print list of bucket items.", value: choices.LIST_ALL_FILES },
        {
          name: "Perform a conditional read.",
          value: choices.CONDITIONAL_READ,
        },
        {
          name: "Perform a conditional copy. These examples use the key name prefix
 defined in ./object_name.json.",
          value: choices.CONDITIONAL_COPY,
        },
        {
          name: "Perform a conditional write. This example use the sample file ./
text02.txt.",
          value: choices.CONDITIONAL_WRITE,
        },
        { name: "Finish the workflow.", value: choices.EXIT },
      ],
    },
  );

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key } = version;
      files.push({ bucket, key: Key });
    }
  }
```

```
  return files;
};

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 * @param {string} key
 */
const getEtag = async (client, bucket, key) => {
  const objectsResponse = await client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    }),
  );
  return objectsResponse.ETag;
};

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
export const replAction = (scenarios, client) =>
  new ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.sourceBucketName,
        state.destinationBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file to use:",
        {
          type: "select",
          choices: files.map((file, index) => ({
            name: `${index + 1}: ${file.bucket}: ${file.key} (Etag: ${
              file.version
```

```
            })`,
            value: index,
          })),
        },
      );
      const condReadOptions = new scenarios.ScenarioInput(
        "selectOption",
        "Which conditional read action would you like to take?",
        {
          type: "select",
          choices: [
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",
            "If-Modified-Since: using yesterday's date. This condition should
succeed.",
            "If-Unmodified-Since: using yesterday's date. This condition should
fail.",
          ],
        },
      );
      const condCopyOptions = new scenarios.ScenarioInput(
        "selectOption",
        "Which conditional copy action would you like to take?",
        {
          type: "select",
          choices: [
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",
            "If-Modified-Since: using yesterday's date. This condition should
succeed.",
            "If-Unmodified-Since: using yesterday's date. This condition should
fail.",
          ],
        },
      );
      const condWriteOptions = new scenarios.ScenarioInput(
        "selectOption",
        "Which conditional write action would you like to take?",
        {
          type: "select",
          choices: [
            "IfNoneMatch condition on the object key: If the key is a duplicate, the
write will fail.",
          ],
```

```
      },
    );

    const { replChoice } = state;

    switch (replChoice) {
      case choices.LIST_ALL_FILES: {
        const files = await getAllFiles(client, [
          state.sourceBucketName,
          state.destinationBucketName,
        ]);
        state.replOutput = files
          .map(
            (file) => `Items in bucket ${file.bucket}: object: ${file.key} `,
          )
          .join("\n");
        break;
      }
      case choices.CONDITIONAL_READ:
        {
          const selectedCondRead = await condReadOptions.handle(state);
          if (
            selectedCondRead ===
            "If-Match: using the object's ETag. This condition should succeed."
          ) {
            const bucket = state.sourceBucketName;
            const key = "file01.txt";
            const ETag = await getEtag(client, bucket, key);

            try {
              await client.send(
                new GetObjectCommand({
                  Bucket: bucket,
                  Key: key,
                  IfMatch: ETag,
                }),
              );
              state.replOutput = `${key} in bucket ${state.sourceBucketName} read
because ETag provided matches the object's ETag.`;
            } catch (err) {
              state.replOutput = `Unable to read object ${key} in bucket
${state.sourceBucketName}: ${err.message}`;
            }
            break;
```

```
            }
            if (
              selectedCondRead ===
              "If-None-Match: using the object's ETag. This condition should fail."
            ) {
              const bucket = state.sourceBucketName;
              const key = "file01.txt";
              const ETag = await getEtag(client, bucket, key);

              try {
                await client.send(
                  new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfNoneMatch: ETag,
                  }),
                );
                state.replOutput = `${key} in ${state.sourceBucketName} was
returned.`;
              } catch (err) {
                state.replOutput = `${key} in ${state.sourceBucketName} was not
read: ${err.message}`;
              }
              break;
            }
            if (
              selectedCondRead ===
              "If-Modified-Since: using yesterday's date. This condition should
succeed."
            ) {
              const date = new Date();
              date.setDate(date.getDate() - 1);

              const bucket = state.sourceBucketName;
              const key = "file01.txt";
              try {
                await client.send(
                  new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfModifiedSince: date,
                  }),
                );
```

```
                state.replOutput = `${key} in bucket ${state.sourceBucketName} read
because it has been created or modified in the last 24 hours.`;
              } catch (err) {
                state.replOutput = `Unable to read object ${key} in bucket
${state.sourceBucketName}: ${err.message}`;
              }
              break;
            }
            if (
              selectedCondRead ===
              "If-Unmodified-Since: using yesterday's date. This condition should
fail."
            ) {
              const bucket = state.sourceBucketName;
              const key = "file01.txt";

              const date = new Date();
              date.setDate(date.getDate() - 1);
              try {
                await client.send(
                  new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfUnmodifiedSince: date,
                  }),
                );
                state.replOutput = `${key} in ${state.sourceBucketName} was read.`;
              } catch (err) {
                state.replOutput = `${key} in ${state.sourceBucketName} was not
read: ${err.message}`;
              }
              break;
            }
          }
          break;
        case choices.CONDITIONAL_COPY: {
          const selectedCondCopy = await condCopyOptions.handle(state);
          if (
            selectedCondCopy ===
            "If-Match: using the object's ETag. This condition should succeed."
          ) {
            const bucket = state.sourceBucketName;
            const key = "file01.txt";
            const ETag = await getEtag(client, bucket, key);
```

```
            const copySource = `${bucket}/${key}`;
            // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
            const name = data.name;
            const copiedKey = `${name}${key}`;
            try {
              await client.send(
                new CopyObjectCommand({
                  CopySource: copySource,
                  Bucket: state.destinationBucketName,
                  Key: copiedKey,
                  CopySourceIfMatch: ETag,
                }),
              );
              state.replOutput = `${key} copied as ${copiedKey} to bucket
${state.destinationBucketName} because ETag provided matches the object's ETag.`;
            } catch (err) {
              state.replOutput = `Unable to copy object ${key} as ${copiedKey} to
bucket ${state.destinationBucketName}: ${err.message}`;
            }
            break;
          }
          if (
            selectedCondCopy ===
            "If-None-Match: using the object's ETag. This condition should fail."
          ) {
            const bucket = state.sourceBucketName;
            const key = "file01.txt";
            const ETag = await getEtag(client, bucket, key);
            const copySource = `${bucket}/${key}`;
            // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
            const name = data.name;
            const copiedKey = `${name}${key}`;

            try {
              await client.send(
                new CopyObjectCommand({
                  CopySource: copySource,
                  Bucket: state.destinationBucketName,
                  Key: copiedKey,
                  CopySourceIfNoneMatch: ETag,
                }),
```

```
            );
            state.replOutput = `${copiedKey} copied to bucket
${state.destinationBucketName}`;
          } catch (err) {
            state.replOutput = `Unable to copy object as ${key} as as ${copiedKey}
to bucket ${state.destinationBucketName}: ${err.message}`;
          }
          break;
        }
        if (
          selectedCondCopy ===
          "If-Modified-Since: using yesterday's date. This condition should
succeed."
        ) {
          const bucket = state.sourceBucketName;
          const key = "file01.txt";
          const copySource = `${bucket}/${key}`;
          // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
          const name = data.name;
          const copiedKey = `${name}${key}`;

          const date = new Date();
          date.setDate(date.getDate() - 1);

          try {
            await client.send(
              new CopyObjectCommand({
                CopySource: copySource,
                Bucket: state.destinationBucketName,
                Key: copiedKey,
                CopySourceIfModifiedSince: date,
              }),
            );
            state.replOutput = `${key} copied as ${copiedKey} to bucket
${state.destinationBucketName} because it has been created or modified in the last
24 hours.`;
          } catch (err) {
            state.replOutput = `Unable to copy object ${key} as ${copiedKey} to
bucket ${state.destinationBucketName} : ${err.message}`;
          }
          break;
        }
        if (
```

```
            selectedCondCopy ===
            "If-Unmodified-Since: using yesterday's date. This condition should
fail."
          ) {
            const bucket = state.sourceBucketName;
            const key = "file01.txt";
            const copySource = `${bucket}/${key}`;
            // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
            const name = data.name;
            const copiedKey = `${name}${key}`;

            const date = new Date();
            date.setDate(date.getDate() - 1);

            try {
              await client.send(
                new CopyObjectCommand({
                  CopySource: copySource,
                  Bucket: state.destinationBucketName,
                  Key: copiedKey,
                  CopySourceIfUnmodifiedSince: date,
                }),
              );
              state.replOutput = `${copiedKey} copied to bucket
${state.destinationBucketName} because it has not been created or modified in the
last 24 hours.`;
            } catch (err) {
              state.replOutput = `Unable to copy object ${key} to bucket
${state.destinationBucketName}: ${err.message}`;
            }
          }
          break;
        }
        case choices.CONDITIONAL_WRITE:
          {
            const selectedCondWrite = await condWriteOptions.handle(state);
            if (
              selectedCondWrite ===
              "IfNoneMatch condition on the object key: If the key is a duplicate,
the write will fail."
            ) {
              // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
```

```
              const key = "text02.txt";
              const __filename = fileURLToPath(import.meta.url);
              const __dirname = dirname(__filename);
              const filePath = path.join(__dirname, "text02.txt");
              try {
                await client.send(
                  new PutObjectCommand({
                    Bucket: `${state.destinationBucketName}`,
                    Key: `${key}`,
                    Body: await readFile(filePath),
                    IfNoneMatch: "*",
                  }),
                );
                state.replOutput = `${key} uploaded to bucket
 ${state.destinationBucketName} because the key is not a duplicate.`;
              } catch (err) {
                state.replOutput = `Unable to upload object to bucket
 ${state.destinationBucketName}:${err.message}`;
              }
              break;
            }
          }
          break;

        default:
          throw new Error(`Invalid replChoice: ${replChoice}`);
      }
    },
    {
      whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new ScenarioOutput("REPL output", (state) => state.replOutput, {
          preformatted: true,
        }),
      },
    },
  );

export { replInput, choices };
```

Détruisez toutes les ressources créées (clean.steps.js).

```
import {
  DeleteObjectCommand,
  DeleteBucketCommand,
  ListObjectVersionsCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { sourceBucketName, destinationBucketName } = state;
    const buckets = [sourceBucketName, destinationBucketName].filter((b) => b);

    for (const bucket of buckets) {
      try {
        let objectsResponse;
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
        );
        for (const version of objectsResponse.Versions || []) {
          const { Key, VersionId } = version;
          try {
            await client.send(
```

```
              new DeleteObjectCommand({
                Bucket: bucket,
                Key,
                VersionId,
              }),
            );
          } catch (err) {
            console.log(`An error occurred: ${err.message} `);
          }
        }
      } catch (e) {
        if (e instanceof Error && e.name === "NoSuchBucket") {
          console.log("Objects and buckets have already been deleted.");
          continue;
        }
        throw e;
      }

      await client.send(new DeleteBucketCommand({ Bucket: bucket }));
      console.log(`Delete for ${bucket} complete.`);
    }
  });

export { confirmCleanup, cleanupAction };
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - CopyObject
  - GetObject
  - PutObject

Charger ou télécharger des fichiers volumineux

L'exemple de code suivant montre comment charger ou télécharger des fichiers volumineux vers et depuis Amazon S3.

Pour plus d'informations, consultez la rubrique Uploading an object using multipart upload (Chargement d'un objet à l'aide du chargement partitionné).

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Chargez un fichier volumineux.

```
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

import {
  ProgressBar,
  logger,
} from "@aws-doc-sdk-examples/lib/utils/util-log.js";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

/**
 * Create a 25MB file and upload it in parts to the specified
 * Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const str = createString();
  const buffer = Buffer.from(str, "utf8");
  const progressBar = new ProgressBar({
    description: `Uploading "${key}" to "${bucketName}"`,
    barLength: 30,
  });

  try {
    const upload = new Upload({
      client: new S3Client({}),
      params: {
        Bucket: bucketName,
        Key: key,
```

```
        Body: buffer,
      },
    });

    upload.on("httpUploadProgress", ({ loaded, total }) => {
      progressBar.update({ current: loaded, total });
    });

    await upload.done();
  } catch (caught) {
    if (caught instanceof Error && caught.name === "AbortError") {
      logger.error(`Multipart upload was aborted. ${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

Téléchargez un fichier volumineux.

```
import { fileURLToPath } from "node:url";
import { GetObjectCommand, NoSuchKey, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream, rmSync } from "node:fs";

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;

export const getObjectRange = ({ bucket, key, start, end }) => {
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

/**
 * @param {string | undefined} contentRange
 */
export const getRangeAndLength = (contentRange) => {
```

```
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: Number.parseInt(start),
    end: Number.parseInt(end),
    length: Number.parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url)),
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
    });
    console.log(`Downloaded bytes ${nextRange.start} to ${nextRange.end}`);

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

/**
 * Download a large object from and Amazon S3 bucket.
 *
 * When downloading a large file, you might want to break it down into
 * smaller pieces. Amazon S3 accepts a Range header to specify the start
 * and end of the byte range to be downloaded.
 *
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
```

```
  try {
    await downloadInChunks({
      bucket: bucketName,
      key: key,
    });
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(`Failed to download object. No such key "${key}".`);
      rmSync(key);
    }
  }
};
```

Exemples sans serveur

Invoquer une fonction lambda à partir d'un déclencheur Amazon S3

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par le chargement d'un objet vers un compartiment S3. La fonction extrait le nom du compartiment S3 et la clé de l'objet à partir du paramètre d'événement et appelle l'API Amazon S3 pour récupérer et consigner le type de contenu de l'objet.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda en utilisant. 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);
    }
};
```

Consommation d'un événement S3 avec Lambda en utilisant. 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);
  }
};
```

SageMaker Exemples d'IA utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec l' SageMaker IA.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Actions
- Scénarios

Mise en route

Bonjour SageMaker AI

L'exemple de code suivant montre comment commencer à utiliser l' SageMaker IA.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  SageMakerClient,
  ListNotebookInstancesCommand,
} from "@aws-sdk/client-sagemaker";

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

export const helloSagemaker = async () => {
  const command = new ListNotebookInstancesCommand({ MaxResults: 5 });

  const response = await client.send(command);
  console.log(
    "Hello Amazon SageMaker! Let's list some of your notebook instances:",
  );

  const instances = response.NotebookInstances || [];

  if (instances.length === 0) {
    console.log(
      "• No notebook instances found. Try creating one in the AWS Management Console
 or with the CreateNotebookInstanceCommand.",
    );
  } else {
    console.log(
      instances
        .map(
          (i) =>
            `• Instance: ${i.NotebookInstanceName}\n  Arn:${
              i.NotebookInstanceArn
            } \n  Creation Date: ${i.CreationTime.toISOString()}`,
        )
        .join("\n"),
```

```
    );
  }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous ListNotebookInstancesà la section Référence des AWS SDK pour JavaScript API.

Actions

`CreatePipeline`

L'exemple de code suivant montre comment utiliser`CreatePipeline`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Fonction qui crée un pipeline d' SageMaker IA à l'aide d'une définition JSON fournie localement.

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
```

```
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../scenarios/features/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
```

```
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}
```

- Pour plus de détails sur l'API, reportez-vous CreatePipelineà la section Référence des AWS SDK pour JavaScript API.

`DeletePipeline`

L'exemple de code suivant montre comment utiliser`DeletePipeline`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Syntaxe permettant de supprimer un pipeline d' SageMaker IA. Ce code fait partie d'une fonction plus large. Reportez-vous à « Créer un pipeline » ou au GitHub référentiel pour plus de contexte.

```
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
```

- Pour plus de détails sur l'API, reportez-vous DeletePipelineà la section Référence des AWS SDK pour JavaScript API.

`DescribePipelineExecution`

L'exemple de code suivant montre comment utiliser`DescribePipelineExecution`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Attendez que l'exécution d'un pipeline d' SageMaker IA réussisse, échoue ou s'arrête.

```
/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  const intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
```

```
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error("Pipeline was forcefully stopped.");
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- Pour plus de détails sur l'API, reportez-vous DescribePipelineExecutionà la section Référence des AWS SDK pour JavaScript API.

`StartPipelineExecution`

L'exemple de code suivant montre comment utiliser`StartPipelineExecution`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Lancez l'exécution d'un pipeline d' SageMaker IA.

```
/**
 * 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,
  };
}
```

- Pour plus de détails sur l'API, reportez-vous StartPipelineExecutionà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Mise en route avec des tâches géospatiales et des pipelines

L’exemple de code suivant illustre comment :

- configurer les ressources d’un pipeline ;
- configurer un pipeline qui exécute une tâche géospatiale ;
- démarrer l’exécution d’un pipeline ;

- surveiller le statut de l'exécution ;
- afficher le résultat du pipeline ;
- nettoyer les ressources.

Pour plus d'informations, consultez Créer et exécuter des SageMaker pipelines à l'aide AWS SDKs de Community.aws.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

L'extrait de fichier suivant contient des fonctions qui utilisent le client SageMaker AI pour gérer un pipeline.

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

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
  GetRoleCommand,
  ListPoliciesCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
  GetFunctionCommand,
} from "@aws-sdk/client-lambda";
```

```
import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-
geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
  GetQueueUrlCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 props
 */
export async function createLambdaExecutionRole({ name, iamClient }) {
  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
```

```
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: { Service: ["lambda.amazonaws.com"] },
            },
          ],
        }),
      }),
    );

  let role = null;

  try {
    const { Role } = await createRole();
    role = Role;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 Amazon SageMaker.
```

```
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: [
          "sqs:ReceiveMessage",
          "sqs:DeleteMessage",
          "sqs:GetQueueAttributes",
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents",
          "sagemaker-geospatial:StartVectorEnrichmentJob",
          "sagemaker-geospatial:GetVectorEnrichmentJob",
          "sagemaker:SendPipelineExecutionStepFailure",
          "sagemaker:SendPipelineExecutionStepSuccess",
          "sagemaker-geospatial:ExportVectorEnrichmentJob",
        ],
        Resource: "*",
      },
      {
        Effect: "Allow",
        // The AWS Lambda function needs permission to pass the pipeline execution
 role to
        // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
 function
        // from elevating privileges. For more information, see:
        // https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_use_passrole.html
        Action: ["iam:PassRole"],
        Resource: `${pipelineExecutionRoleArn}`,
        Condition: {
          StringEquals: {
            "iam:PassedToService": [
              "sagemaker.amazonaws.com",
              "sagemaker-geospatial.amazonaws.com",
```

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

  const createPolicy = () =>
    iamClient.send(
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
        PolicyName: name,
      }),
    );

  let policy = null;

  try {
    const { Policy } = await createPolicy();
    policy = Policy;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
      if (Policies) {
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}
```

```
/**
 * Attach an AWS IAM policy to an AWS IAM role.
 * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-
iam').IAMClient}} props
 */
export async function attachPolicy({ roleName, policyArn, iamClient }) {
  const attachPolicyCommand = new AttachRolePolicyCommand({
    RoleName: roleName,
    PolicyArn: policyArn,
  });

  await iamClient.send(attachPolicyCommand);
  return {
    cleanUp: async () => {
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: roleName,
          PolicyArn: policyArn,
        }),
      );
    },
  };
}

/**
 * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon
 SageMaker Geospatial clients
 * in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
 Amazon SageMaker
 * Geospatial client wasn't introduced until v3.221.0.
 * @param {{ name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient }} props
 */
export async function createLambdaLayer({ name, lambdaClient }) {
  const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;
  const { LayerVersionArn, Version } = await lambdaClient.send(
    new PublishLayerVersionCommand({
      LayerName: name,
      Content: {
        ZipFile: Uint8Array.from(readFileSync(layerPath)),
      },
    }),
  );
```

```
  return {
    versionArn: LayerVersionArn,
    version: Version,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteLayerVersionCommand({
          LayerName: name,
          VersionNumber: Version,
        }),
      );
    },
  };
}

/**
 * Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
 pipeline
 * execution steps.
 * @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient, layerVersionArn: string}} props
 */
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
  layerVersionArn,
}) {
  const lambdaPath = `${dirnameFromMetaUrl(
    import.meta.url,
  )}lambda/dist/index.mjs.zip`;

  // If a function of the same name already exists, return that
  // function's ARN instead. By default this is
  // "sagemaker-wkflw-lambda-function", so collisions are
  // unlikely.
  const createFunction = async () => {
    try {
      return await lambdaClient.send(
        new CreateFunctionCommand({
          Code: {
            ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
          },
          Runtime: Runtime.nodejs18x,
```

```
          Handler: "index.handler",
          Layers: [layerVersionArn],
          FunctionName: name,
          Role: roleArn,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "ResourceConflictException"
      ) {
        const { Configuration } = await lambdaClient.send(
          new GetFunctionCommand({ FunctionName: name }),
        );
        return Configuration;
      }
      throw caught;
    }
  };

  // Function creation fails if the Role is not ready. This retries
  // function creation until it succeeds or it times out.
  const { FunctionArn } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    createFunction,
  );

  return {
    arn: FunctionArn,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteFunctionCommand({ FunctionName: name }),
      );
    },
  };
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
 props
```

```
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
  const s3Path = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../../scenarios/features/sagemaker_pipelines/resources/
latlongtest.csv`;

  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "input/sample_data.csv",
      Body: readFileSync(s3Path),
    }),
  );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient, wait:
 (ms: number) => Promise<void>}} props
 */
export async function createSagemakerRole({ name, iamClient, wait }) {
  let role = null;

  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: {
                Service: [
                  "sagemaker.amazonaws.com",
                  "sagemaker-geospatial.amazonaws.com",
                ],
              },
            },
          ],
        }),
      }),
```

```
    );

  try {
    const { Role } = await createRole();
    role = Role;
    // Wait for the role to be ready.
    await wait(10);
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 messages to
 * the Amazon SQS queue.
 * @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
 import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
 */
export async function createSagemakerExecutionPolicy({
  sqsQueueArn,
  lambdaArn,
  iamClient,
  name,
  s3BucketName,
}) {
```

```
const policyConfig = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: ["lambda:InvokeFunction"],
      Resource: lambdaArn,
    },
    {
      Effect: "Allow",
      Action: ["s3:*"],
      Resource: [
        `arn:aws:s3:::${s3BucketName}`,
        `arn:aws:s3:::${s3BucketName}/*`,
      ],
    },
    {
      Effect: "Allow",
      Action: ["sqs:SendMessage"],
      Resource: sqsQueueArn,
    },
  ],
};

const createPolicy = () =>
  iamClient.send(
    new CreatePolicyCommand({
      PolicyDocument: JSON.stringify(policyConfig),
      PolicyName: name,
    }),
  );

let policy = null;

try {
  const { Policy } = await createPolicy();
  policy = Policy;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
    if (Policies) {
```

```
        policy = Policies.find((p) => p.PolicyName === name);
      } else {
        throw new Error("No policies found.");
      }
    } else {
      throw caught;
    }
  }

  return {
    arn: policy?.Arn,
    policyConfig,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
    },
  };
}

/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../scenarios/features/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);
```

```
  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
```

```
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
  const createSqsQueue = () =>
    sqsClient.send(
      new CreateQueueCommand({
        QueueName: name,
        Attributes: {
          DelaySeconds: "5",
          ReceiveMessageWaitTimeSeconds: "5",
          VisibilityTimeout: "300",
        },
      }),
    );

  let queueUrl = null;
  try {
    const { QueueUrl } = await createSqsQueue();
    queueUrl = QueueUrl;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "QueueNameExists") {
      const { QueueUrl } = await sqsClient.send(
        new GetQueueUrlCommand({ QueueName: name }),
      );
      queueUrl = QueueUrl;
    } else {
      throw caught;
    }
  }

  const { Attributes } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    () =>
      sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: queueUrl,
          AttributeNames: ["QueueArn"],
        }),
      ),
  );

  return {
    queueUrl,
```

```
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl: queueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{
 *   paginateListEventSourceMappings: () => Generator<import('@aws-sdk/client-
lambda').ListEventSourceMappingsCommandOutput>,
 *   lambdaName: string,
 *   queueArn: string,
 *   lambdaClient: import('@aws-sdk/client-lambda').LambdaClient}} props
 */
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
  paginateListEventSourceMappings,
}) {
  let uuid = null;
  const createEvenSourceMapping = () =>
    lambdaClient.send(
      new CreateEventSourceMappingCommand({
        EventSourceArn: queueArn,
        FunctionName: lambdaName,
      }),
    );

  try {
    const { UUID } = await createEvenSourceMapping();
    uuid = UUID;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceConflictException"
    ) {
      const paginator = paginateListEventSourceMappings(
        { client: lambdaClient },
        {},
      );
```

```
      /**
       * @type {import('@aws-sdk/client-lambda').EventSourceMappingConfiguration[]}
       */
      const eventSourceMappings = [];
      for await (const page of paginator) {
        eventSourceMappings.concat(page.EventSourceMappings || []);
      }

      const { Configuration } = await lambdaClient.send(
        new GetFunctionCommand({ FunctionName: lambdaName }),
      );

      uuid = eventSourceMappings.find(
        (mapping) =>
          mapping.EventSourceArn === queueArn &&
          mapping.FunctionArn === Configuration.FunctionArn,
      ).UUID;
    } else {
      throw caught;
    }
  }

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
          UUID: uuid,
        }),
      );
    },
  };
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{
 *   s3Client: import('@aws-sdk/client-s3').S3Client,
 *   name: string,
 *   paginateListObjectsV2: () => Generator<import('@aws-sdk/client-
s3').ListObjectsCommandOutput>
 * }} props
 */
export async function createS3Bucket({
```

```
  name,
  s3Client,
  paginateListObjectsV2,
}) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(
        { client: s3Client },
        { Bucket: name },
      );
      for await (const page of paginator) {
        const objects = page.Contents;
        if (objects) {
          for (const object of objects) {
            await s3Client.send(
              new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
            );
          }
        }
      }
      await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
    },
  };
}

/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
```

```
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
```

```
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}

/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  const intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
```

```
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error("Pipeline was forcefully stopped.");
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-
s3').S3Client}} param0
 */
export async function getObject({ bucket, s3Client }) {
  const prefix = "output/";
  const { Contents } = await s3Client.send(
    new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
  );

  if (!Contents.length) {
    throw new Error("No objects found in bucket.");
  }

  // Find the CSV file.
  const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

  if (!outputObject) {
    throw new Error(`No CSV file found in bucket with the prefix "${prefix}".`);
```

```
  }

  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: outputObject.Key,
    }),
  );

  return Body.transformToString();
}
```

Cette fonction est un extrait d'un fichier qui utilise les fonctions de bibliothèque précédentes pour configurer un pipeline d' SageMaker IA, l'exécuter et supprimer toutes les ressources créées.

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

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution
  - UpdatePipeline

Exemples de Secrets Manager utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) with Secrets Manager.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`GetSecretValue`

L'exemple de code suivant montre comment utiliser`GetSecretValue`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";

export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: secretName,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
  //   CreatedDate: 2023-08-08T19:29:51.294Z,
  //   Name: 'binary-secret-3873048',
  //   SecretBinary: Uint8Array(11) [
  //      98, 105, 110, 97, 114,
```

```
  //     121,  32, 100, 97, 116,
  //      97
  //   ],
  //   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
  //   VersionStages: [ 'AWSCURRENT' ]
  // }

  if (response.SecretString) {
    return response.SecretString;
  }

  if (response.SecretBinary) {
    return response.SecretBinary;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous GetSecretValueà la section Référence des AWS SDK pour JavaScript API.

Exemples Amazon SES utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la AWS SDK pour JavaScript version (v3) avec Amazon SES.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions
- Scénarios

Actions

`CreateReceiptFilter`

L'exemple de code suivant montre comment utiliser`CreateReceiptFilter`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
 address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
 addressesOptions.
      },
      /*
        The name of the IP address filter. Only ASCII letters, numbers, underscores,
 or dashes.
        Must be less than 64 characters and start and end with a letter or number.
       */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");
```

```
const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });

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

- Pour plus de détails sur l'API, reportez-vous CreateReceiptFilterà la section Référence des AWS SDK pour JavaScript API.

`CreateReceiptRule`

L'exemple de code suivant montre comment utiliser`CreateReceiptRule`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const RULE_SET_NAME = getUniqueName("RuleSetName");
```

```
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
  });
};

const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateReceiptRuleà la section Référence des AWS SDK pour JavaScript API.

`CreateReceiptRuleSet`

L'exemple de code suivant montre comment utiliser`CreateReceiptRuleSet`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createCreateReceiptRuleSetCommand = (ruleSetName) => {
  return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
};

const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateReceiptRuleSetà la section Référence des AWS SDK pour JavaScript API.

`CreateTemplate`

L'exemple de code suivant montre comment utiliser`CreateTemplate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

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

```
};

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

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

- Pour plus de détails sur l'API, reportez-vous CreateTemplateà la section Référence des AWS SDK pour JavaScript API.

`DeleteIdentity`

L'exemple de code suivant montre comment utiliser`DeleteIdentity`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

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

```
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

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

- Pour plus de détails sur l'API, reportez-vous DeleteIdentityà la section Référence des AWS SDK pour JavaScript API.

`DeleteReceiptFilter`

L'exemple de code suivant montre comment utiliser`DeleteReceiptFilter`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
```

```
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteReceiptFilterà la section Référence des AWS SDK pour JavaScript API.

`DeleteReceiptRule`

L'exemple de code suivant montre comment utiliser`DeleteReceiptRule`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleCommand = () => {
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
```

```
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteReceiptRuleà la section Référence des AWS SDK pour JavaScript API.

`DeleteReceiptRuleSet`

L'exemple de code suivant montre comment utiliser`DeleteReceiptRuleSet`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};

const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
```

```
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteReceiptRuleSetà la section Référence des AWS SDK pour JavaScript API.

`DeleteTemplate`

L'exemple de code suivant montre comment utiliser`DeleteTemplate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const TEMPLATE_NAME = getUniqueName("TemplateName");

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

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

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

- Pour plus de détails sur l'API, reportez-vous DeleteTemplateà la section Référence des AWS SDK pour JavaScript API.

`GetTemplate`

L'exemple de code suivant montre comment utiliser`GetTemplate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const TEMPLATE_NAME = getUniqueName("TemplateName");

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

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

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

- Pour plus de détails sur l'API, reportez-vous GetTemplateà la section Référence des AWS SDK pour JavaScript API.

`ListIdentities`

L'exemple de code suivant montre comment utiliser`ListIdentities`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ListIdentitiesà la section Référence des AWS SDK pour JavaScript API.

`ListReceiptFilters`

L'exemple de code suivant montre comment utiliser`ListReceiptFilters`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
};
```

- Pour plus de détails sur l'API, reportez-vous ListReceiptFiltersà la section Référence des AWS SDK pour JavaScript API.

`ListTemplates`

L'exemple de code suivant montre comment utiliser`ListTemplates`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });
```

```
const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ListTemplatesà la section Référence des AWS SDK pour JavaScript API.

`SendBulkTemplatedEmail`

L'exemple de code suivant montre comment utiliser`SendBulkTemplatedEmail`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

/**
```

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

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

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

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

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

- Pour plus de détails sur l'API, reportez-vous SendBulkTemplatedEmailà la section Référence des AWS SDK pour JavaScript API.

`SendEmail`

L'exemple de code suivant montre comment utiliser`SendEmail`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
```

```
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

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

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

```
  }
};
```

- Pour plus de détails sur l'API, reportez-vous SendEmailà la section Référence des AWS SDK pour JavaScript API.

`SendRawEmail`

L'exemple de code suivant montre comment utiliser`SendRawEmail`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez nodemailer pour envoyer un e-mail avec une pièce jointe.

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

- Pour plus de détails sur l'API, reportez-vous SendRawEmailà la section Référence des AWS SDK pour JavaScript API.

`SendTemplatedEmail`

L'exemple de code suivant montre comment utiliser`SendTemplatedEmail`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
```

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

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

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

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
 party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
```

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

- Pour plus de détails sur l'API, reportez-vous SendTemplatedEmailà la section Référence des AWS SDK pour JavaScript API.

`UpdateTemplate`

L'exemple de code suivant montre comment utiliser`UpdateTemplate`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

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

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

```
  });
};

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

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

- Pour plus de détails sur l'API, reportez-vous UpdateTemplateà la section Référence des AWS SDK pour JavaScript API.

`VerifyDomainIdentity`

L'exemple de code suivant montre comment utiliser`VerifyDomainIdentity`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
```

```
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous VerifyDomainIdentityà la section Référence des AWS SDK pour JavaScript API.

`VerifyEmailIdentity`

L'exemple de code suivant montre comment utiliser`VerifyEmailIdentity`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
```

```
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous VerifyEmailIdentityà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer une application de streaming Amazon Transcribe

L'exemple de code suivant montre comment créer une application qui enregistre, transcrit et traduit de l'audio en direct en temps réel, et envoie les résultats par e-mail.

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Transcribe afin de créer une application qui enregistre, transcrit et traduit de l'audio en direct en temps réel, et envoie les résultats par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Créer un outil de suivi des éléments de travail sans serveur Aurora

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une base de données Amazon Aurora sans serveur et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK pour JavaScript (v3) pour créer une application Web qui suit les éléments de travail dans une base de données Amazon Aurora et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise un front end créé avec React.js pour interagir avec un backend Express Node.js.

- Intégrez une application Web React.js à Services AWS.
- Lister, ajouter et mettre à jour des éléments dans une table Aurora.
- Envoyez un rapport par e-mail sur les éléments de travail filtrés en utilisant Amazon SES.
- Déployez et gérez des exemples de ressources à l'aide du AWS CloudFormation script inclus.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Aurora
- Amazon RDS
- Services de données Amazon RDS
- Amazon SES

Détecter des objets dans des images

L'exemple de code suivant montre comment créer une application qui utilise Amazon Rekognition afin de détecter des objets par catégorie dans des images.

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Rekognition AWS SDK pour JavaScript pour créer une application qui utilise Amazon Rekognition pour identifier les objets par catégorie dans des images situées dans un compartiment Amazon Simple Storage Service (Amazon S3). L'application envoie

à l'administrateur une notification par e-mail contenant les résultats à l'aide d'Amazon Simple Email Service (Amazon SES).

Découvrez comment :

- Créer un utilisateur non authentifié à l'aide d'Amazon Cognito.
- Analyser les images à la recherche d'objets à l'aide d'Amazon Rekognition.
- Vérifier une adresse e-mail pour Amazon SES.
- Envoyer une notification par e-mail à l'aide d'Amazon SES.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Rekognition
- Amazon S3
- Amazon SES

Exemples Amazon SNS utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon SNS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Actions

- Scénarios
- Exemples sans serveur

Mise en route

Hello Amazon SNS

L'exemple de code suivant montre comment commencer à utiliser Amazon SNS.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Initialisez un client SNS et répertoriez les rubriques figurant dans votre compte.

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

- Pour plus de détails sur l'API, reportez-vous ListTopicsà la section Référence des AWS SDK pour JavaScript API.

Actions

`CheckIfPhoneNumberIsOptedOut`

L'exemple de code suivant montre comment utiliser`CheckIfPhoneNumberIsOptedOut`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

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

```
export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous CheckIfPhoneNumberIsOptedOutà la section Référence des AWS SDK pour JavaScript API.

`ConfirmSubscription`

L'exemple de code suivant montre comment utiliser`ConfirmSubscription`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
```

```
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous ConfirmSubscriptionà la section Référence des AWS SDK pour JavaScript API.

`CreateTopic`

L'exemple de code suivant montre comment utiliser`CreateTopic`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {string} topicName - The name of the topic to create.
```

```
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous CreateTopicà la section Référence des AWS SDK pour JavaScript API.

`DeleteTopic`

L'exemple de code suivant montre comment utiliser`DeleteTopic`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous DeleteTopicà la section Référence des AWS SDK pour JavaScript API.

`GetSMSAttributes`

L'exemple de code suivant montre comment utiliser`GetSMSAttributes`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
```

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

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, voir Get SMSAttributes in AWS SDK pour JavaScript API Reference.

`GetTopicAttributes`

L'exemple de code suivant montre comment utiliser`GetTopicAttributes`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
```

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

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous GetTopicAttributesà la section Référence des AWS SDK pour JavaScript API.

`ListSubscriptions`

L'exemple de code suivant montre comment utiliser`ListSubscriptions`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous ListSubscriptionsà la section Référence des AWS SDK pour JavaScript API.

`ListTopics`

L'exemple de code suivant montre comment utiliser`ListTopics`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous ListTopicsà la section Référence des AWS SDK pour JavaScript API.

`Publish`

L'exemple de code suivant montre comment utiliser`Publish`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                  if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
```

```
};
```

Publiez un message dans une rubrique avec des options de groupe, de duplication et d'attribut.

```
  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
```

```
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus d'informations sur l'API, consultez Publier dans AWS SDK pour JavaScript Référence de l'API.

`SetSMSAttributes`

L'exemple de code suivant montre comment utiliser`SetSMSAttributes`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, voir Set SMSAttributes in AWS SDK pour JavaScript API Reference.

`SetTopicAttributes`

L'exemple de code suivant montre comment utiliser`SetTopicAttributes`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
```

```
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous SetTopicAttributesà la section Référence des AWS SDK pour JavaScript API.

`Subscribe`

L'exemple de code suivant montre comment utiliser`Subscribe`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

Abonnez une application mobile à une rubrique.

```
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;
};
```

Abonnez une fonction Lambda à une rubrique.

```
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;
};
```

Abonnez une file d'attente SQS à une rubrique.

```
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;
};
```

Abonnez-vous avec un filtre à une rubrique.

```
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;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour de plus amples informations sur l'API, consultez S'abonner dans AWS SDK pour JavaScript Référence de l'API.

`Unsubscribe`

L'exemple de code suivant montre comment utiliser`Unsubscribe`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client dans un module séparé et exportez-le.

```
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({});
```

Importez le kit SDK et les modules client et appelez l'API.

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

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Pour plus d'informations, consultez le AWS SDK pour JavaScript Guide du développeur.
- Pour de plus amples informations sur l'API, consultez Se désabonner dans Référence de l'API AWS SDK pour JavaScript .

Scénarios

Créer une application pour soumettre des données à une table DynamoDB

L'exemple de code suivant montre comment créer une application qui soumet des données à une table Amazon DynamoDB et vous avertit lorsqu'un utilisateur met à jour la table.

SDK pour JavaScript (v3)

Cet exemple montre comment créer une application qui permet aux utilisateurs de soumettre des données à une table Amazon DynamoDB et d'envoyer un message texte à l'administrateur à l'aide d'Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SNS

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour JavaScript (v3)

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- API Gateway

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

Créer une application Amazon Textract Explorer

L'exemple de code suivant montre comment explorer la sortie Amazon Textract via une application interactive.

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK pour JavaScript pour créer une application React qui utilise Amazon Textract pour extraire des données d'une image de document et les afficher sur une page Web interactive. Cet exemple s'exécute dans un navigateur Web et nécessite une identité Amazon Cognito authentifiée pour les informations d'identification. Il utilise Amazon Simple Storage Service (Amazon S3) pour le stockage et, pour les notifications, il interroge une file d'attente Amazon Simple Queue Service (Amazon SQS) abonnée à une rubrique Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Publier des messages dans des files d'attente

L'exemple de code suivant illustre comment :

- Créer une rubrique (FIFO ou non FIFO).
- Abonner plusieurs files d'attente à la rubrique avec la possibilité d'appliquer un filtre.

- Publier des messages dans la rubrique.
- Interroger les files d'attente pour les messages reçus.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Il s'agit du point d'entrée de ce scénario.

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

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Le code précédent fournit les dépendances nécessaires et démarre le scénario. La section suivante contient l'essentiel de l'exemple.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
```

```
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../../libs/prompter.js').Prompter} prompter
   * @param {import('../../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
    await this.logger.log(MESSAGES.snsFifoDescription);
    this.isFifo = await this.prompter.confirm({
      message: MESSAGES.snsFifoPrompt,
    });
```

```
    if (this.isFifo) {
      this.logger.logSeparator(MESSAGES.headerDedup);
      await this.logger.log(MESSAGES.deduplicationNotice);
      await this.logger.log(MESSAGES.deduplicationDescription);
      this.autoDedup = await this.prompter.confirm({
        message: MESSAGES.deduplicationPrompt,
      });
    }
  }

  async createTopic() {
    await this.logger.log(MESSAGES.creatingTopics);
    this.topicName = await this.prompter.input({
      message: MESSAGES.topicNamePrompt,
    });
    if (this.isFifo) {
      this.topicName += ".fifo";
      this.logger.logSeparator(MESSAGES.headerFifoNaming);
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    const maxQueues = 2;
```

```
    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }
```

```
  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
```

```
        }),
      );
      queue.policy = policy;
    } else {
      await this.logger.log(
        MESSAGES.policyNotAttachedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
    }
  }
}

async subscribeQueuesToTopic() {
  for (const [index, queue] of this.queues.entries()) {
    /**
     * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
     */
    const subscribeParams = {
      TopicArn: this.topicArn,
      Protocol: "sqs",
      Endpoint: queue.queueArn,
    };
    let tones = [];

    if (this.isFifo) {
      if (index === 0) {
        await this.logger.log(MESSAGES.fifoFilterNotice);
      }
      tones = await this.prompter.checkbox({
        message: MESSAGES.fifoFilterSelect.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
        choices: toneChoices,
      });

      if (tones.length) {
        subscribeParams.Attributes = {
          FilterPolicyScope: "MessageAttributes",
          FilterPolicy: JSON.stringify({
            tone: tones,
          }),
```

```
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
```

```
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
```

```
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
```

```
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Utiliser API Gateway pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par Amazon API Gateway.

SDK pour JavaScript (v3)

Montre comment créer une AWS Lambda fonction à l'aide de l'API JavaScript d'exécution Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une fonction Lambda invoquée par Amazon API Gateway qui analyse une table Amazon DynamoDB à la recherche d'anniversaires professionnels et utilise Amazon Simple Notification Service (Amazon SNS) pour envoyer un message texte à vos employés qui les félicitent à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Utilisent des événements planifiés pour appeler une fonction Lambda

L'exemple de code suivant montre comment créer une AWS Lambda fonction invoquée par un événement EventBridge planifié par Amazon.

SDK pour JavaScript (v3)

Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d’anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK pour JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Exemples sans serveur

Invocation d'une fonction lambda à partir d'un déclencheur Amazon SNS

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d'une rubrique SNS. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement SNS avec JavaScript Lambda en utilisant.

```
// 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;
  }
}
```

Consommation d'un événement SNS avec TypeScript Lambda en utilisant.

```
// 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;
  }
}
```

Exemples Amazon SQS utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon SQS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d’un même service ou combinés à d’autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l’exécution du code en contexte.

Rubriques

Mise en route

Bonjour Amazon SQS

L'exemple de code suivant montre comment commencer à utiliser Amazon SQS.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Initialisez un client Amazon SQS et répertoriez les files d'attente.

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

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour JavaScript API.

Actions

`ChangeMessageVisibility`

L'exemple de code suivant montre comment utiliser`ChangeMessageVisibility`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez un message Amazon SQS et modifiez la visibilité du délai d'expiration.

```
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;
};
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des AWS SDK pour JavaScript API.

`CreateQueue`

L'exemple de code suivant montre comment utiliser`CreateQueue`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une file d'attente standard Amazon SQS.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";
```

```
const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

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

Créez une file d'attente Amazon SQS avec une attente active de longue durée.

```
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;
```

```
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section Référence des AWS SDK pour JavaScript API.

`DeleteMessage`

L'exemple de code suivant montre comment utiliser`DeleteMessage`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez et supprimez des messages Amazon SQS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
```

```
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageà la section Référence des AWS SDK pour JavaScript API.

`DeleteMessageBatch`

L'exemple de code suivant montre comment utiliser`DeleteMessageBatch`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
```

```
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageBatchà la section Référence des AWS SDK pour JavaScript API.

`DeleteQueue`

L'exemple de code suivant montre comment utiliser`DeleteQueue`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez une file d'attente Amazon SQS.

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });
```

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

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteQueueà la section Référence des AWS SDK pour JavaScript API.

`GetQueueAttributes`

L'exemple de code suivant montre comment utiliser`GetQueueAttributes`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
```

```
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous GetQueueAttributesà la section Référence des AWS SDK pour JavaScript API.

`GetQueueUrl`

L'exemple de code suivant montre comment utiliser`GetQueueUrl`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Récupérez l'URL d'une file d'attente Amazon SQS.

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

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

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, reportez-vous GetQueueUrlà la section Référence des AWS SDK pour JavaScript API.

`ListQueues`

L'exemple de code suivant montre comment utiliser`ListQueues`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez vos files d'attente Amazon SQS.

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    for (const url of urls) {
      console.log(url);
    }
  }

  return urls;
};
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour JavaScript API.

`ReceiveMessage`

L'exemple de code suivant montre comment utiliser`ReceiveMessage`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez un message d'une file d'attente SQS.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }
```

```
  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

Recevez un message d'une file d'attente Amazon SQS grâce à la prise en charge de l'attente active de longue durée.

```
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;
};
```

- Pour plus de détails sur l'API, reportez-vous ReceiveMessageà la section Référence des AWS SDK pour JavaScript API.

`SendMessage`

L'exemple de code suivant montre comment utiliser`SendMessage`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message à une file d'attente Amazon SQS.

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
```

```
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
    MessageBody:
      "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  });

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

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous SendMessageà la section Référence des AWS SDK pour JavaScript API.

`SetQueueAttributes`

L'exemple de code suivant montre comment utiliser`SetQueueAttributes`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
```

```
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

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

Configurez une file d'attente Amazon SQS pour utiliser une attente active de longue durée.

```
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;
};
```

Configurer une file d'attente de lettres mortes.

```
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;
};
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer une application Amazon Textract Explorer

L’exemple de code suivant montre comment explorer la sortie Amazon Textract via une application interactive.

SDK pour JavaScript (v3)

> Montre comment utiliser le AWS SDK pour JavaScript pour créer une application React qui utilise Amazon Textract pour extraire des données d'une image de document et les afficher sur une page Web interactive. Cet exemple s’exécute dans un navigateur Web et nécessite une identité Amazon Cognito authentifiée pour les informations d’identification. Il utilise Amazon Simple Storage Service (Amazon S3) pour le stockage et, pour les notifications, il interroge une file

d'attente Amazon Simple Queue Service (Amazon SQS) abonnée à une rubrique Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Publier des messages dans des files d'attente

L'exemple de code suivant illustre comment :

- Créer une rubrique (FIFO ou non FIFO).
- Abonner plusieurs files d'attente à la rubrique avec la possibilité d'appliquer un filtre.
- Publier des messages dans la rubrique.
- Interroger les files d'attente pour les messages reçus.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Il s'agit du point d'entrée de ce scénario.

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

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
```

```
export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Le code précédent fournit les dépendances nécessaires et démarre le scénario. La section suivante contient l'essentiel de l'exemple.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;
```

```
/**
 * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
 * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
 * @param {import('../../libs/prompter.js').Prompter} prompter
 * @param {import('../../libs/logger.js').Logger} logger
 */
constructor(snsClient, sqsClient, prompter, logger) {
  this.snsClient = snsClient;
  this.sqsClient = sqsClient;
  this.prompter = prompter;
  this.logger = logger;
}

async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
  }
```

```
    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    const maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
```

```
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
```

```
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
```

```
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
```

```
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
```

```
              tone: {
                DataType: "String.Array",
                StringValue: JSON.stringify(choices),
              },
            },
          }
        : {}),
    }),
  );

  const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
  });

  if (publishAnother) {
    await this.publishMessages();
  }
}

async receiveAndDeleteMessages() {
  for (const queue of this.queues) {
    const { Messages } = await this.sqsClient.send(
      new ReceiveMessageCommand({
        QueueUrl: queue.queueUrl,
      }),
    );

    if (Messages) {
      await this.logger.log(
        MESSAGES.messagesReceivedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
      console.log(Messages);

      await this.sqsClient.send(
        new DeleteMessageBatchCommand({
          QueueUrl: queue.queueUrl,
          Entries: Messages.map((message) => ({
            Id: message.MessageId,
            ReceiptHandle: message.ReceiptHandle,
          })),
        }),
```

```
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
```

```
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Exemples sans serveur

Invoquer une fonction Lambda à partir d'un déclencheur Amazon SQS

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d'une file d'attente SQS. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement SQS avec JavaScript Lambda en utilisant.

```
// 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;
  }
}
```

Consommation d'un événement SQS avec TypeScript Lambda en utilisant.

```
// 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;
  }
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'une file d'attente SQS. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot SQS avec JavaScript Lambda à l'aide de.

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

Signaler les défaillances d'éléments de lot SQS avec TypeScript Lambda à l'aide de.

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

Exemples de Step Functions utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) with Step Functions.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`StartExecution`

L'exemple de code suivant montre comment utiliser`StartExecution`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
  return response;
}

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- Pour plus de détails sur l'API, reportez-vous StartExecutionà la section Référence des AWS SDK pour JavaScript API.

AWS STS exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec AWS STS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Actions

`AssumeRole`

L'exemple de code suivant montre comment utiliser`AssumeRole`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

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

Assumez le rôle IAM.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous AssumeRoleà la section Référence des AWS SDK pour JavaScript API.

Support exemples d'utilisation du SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour JavaScript (v3) avec Support.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour Support

L'exemple de code suivant montre comment démarrer avec Support.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Appelez « main() » pour exécuter l'exemple.

```
import {
  DescribeServicesCommand,
  SupportClient,
} from "@aws-sdk/client-support";

// Change the value of 'region' to your preferred AWS Region.
const client = new SupportClient({ region: "us-east-1" });

const getServiceCount = async () => {
  try {
    const { services } = await client.send(new DescribeServicesCommand({}));
    return services.length;
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
```

```
      );
    }
    throw err;
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeServicesà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Obtenez et affichez les services disponibles et les niveaux de gravité des dossiers.
- Créez un dossier de support en utilisant un service, une catégorie et un niveau de gravité sélectionnés.
- Obtenez et affichez une liste des dossiers ouverts pour la journée en cours.
- Ajoutez un ensemble de pièces jointes et une communication au nouveau dossier.
- Décrivez la nouvelle pièce jointe et la nouvelle communication relatives au dossier.
- Résolvez le dossier.
- Obtenez et affichez la liste des dossiers ouverts pour la journée en cours.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif dans le terminal.

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import * as inquirer from "@inquirer/prompts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};

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

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});

  try {
    await client.send(command);
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
```

```
    }
    throw err;
  }
};

/**
 * Select a service from the list returned from DescribeServices.
 */
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const selectedService = await inquirer.select({
    message:
      "Select a service. Your support case will be created for this service. The
 list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
};

/**
 * @param {{ categories: import('@aws-sdk/client-support').Category[]}} service
 */
export const getCategory = async (service) => {
  const selectedCategory = await inquirer.select({
    message: "Select a category.",
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const selectedSeverityLevel = await inquirer.select({
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

/**
 * Create a new support case
 * @param {{
```

```
 *  selectedService: import('@aws-sdk/client-support').Service
 *  selectedCategory: import('@aws-sdk/client-support').Category
 *  selectedSeverityLevel: import('@aws-sdk/client-support').SeverityLevel
 * }} selections
 * @returns
 */
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodaysOpenCases = async () => {
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });

  const { cases } = await client.send(command);

  if (cases.length === 0) {
    throw new Error(
      "Unexpected number of cases. Expected more than 0 open cases.",
    );
  }
  return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
  const command = new AddAttachmentsToSetCommand({
```

```
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
  });
  const { communications } = await client.send(command);
  return communications;
};

/**
 * @param {import('@aws-sdk/client-support').Communication[]} communications
 */
export const getFirstAttachment = (communications) => {
  const firstCommWithAttachment = communications.find(
    (c) => c.attachmentSet.length > 0,
  );
  return firstCommWithAttachment?.attachmentSet[0].attachmentId;
};

// Get an attachment.
export const getAttachment = async (attachmentId) => {
  const command = new DescribeAttachmentCommand({
    attachmentId,
  });
```

```
  const { attachment } = await client.send(command);
  return attachment;
};

// Resolve the case matching the given case ID.
export const resolveCase = async (caseId) => {
  const shouldResolve = await inquirer.confirm({
    message: `Do you want to resolve ${caseId}?`,
  });

  if (shouldResolve) {
    const command = new ResolveCaseCommand({
      caseId: caseId,
    });

    await client.send(command);
    return true;
  }
  return false;
};

/**
 * Find a specific case in the list of provided cases by case ID.
 * If the case is not found, and the results are paginated, continue
 * paging through the results.
 * @param {{
 *   caseId: string,
 *   cases: import('@aws-sdk/client-support').CaseDetails[]
 *   nextToken: string
 * }} options
 * @returns
 */
export const findCase = async ({ caseId, cases, nextToken }) => {
  const foundCase = cases.find((c) => c.caseId === caseId);

  if (foundCase) {
    return foundCase;
  }

  if (nextToken) {
    const response = await client.send(
      new DescribeCasesCommand({
        nextToken,
        includeResolvedCases: true,
```

```
      }),
    );
    return findCase({
      caseId,
      cases: response.cases,
      nextToken: response.nextToken,
    });
  }

  throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
 select one.
    const selectedCategory = await getCategory(selectedService);

    // Provide the severity available severity levels for the account and prompt the
 user to select one.
    const selectedSeverityLevel = await getSeverityLevel();
```

```
  // Create a support case.
  console.log("\nCreating a support case.");
  caseId = await createCase({
    selectedService,
    selectedCategory,
    selectedSeverityLevel,
  });
  console.log(`Support case created: ${caseId}`);

  // Display a list of open support cases created today.
  const todaysOpenCases = await retry(
    { intervalInMs: 1000, maxRetries: 15 },
    getTodaysOpenCases,
  );
  console.log(
    `\nOpen support cases created today: ${todaysOpenCases.length}`,
  );
  console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

  // Create an attachment set.
  console.log("\nCreating an attachment set.");
  const attachmentSetId = await createAttachmentSet();
  console.log(`Attachment set created: ${attachmentSetId}`);

  // Add the attachment set to the support case.
  console.log(`\nAdding attachment set to ${caseId}`);
  await linkAttachmentSetToCase(attachmentSetId, caseId);
  console.log(`Attachment set added to ${caseId}`);

  // List the communications for a support case.
  console.log(`\nListing communications for ${caseId}`);
  const communications = await getCommunications(caseId);
  console.log(
    communications
      .map(
        (c) =>
          `Communication created on ${c.timeCreated}. Has
${c.attachmentSet.length} attachments.`,
      )
      .join("\n"),
  );

  // Describe the first attachment.
```

```
    console.log(`\nDescribing attachment ${attachmentSetId}`);
    const attachmentId = getFirstAttachment(communications);
    const attachment = await getAttachment(attachmentId);
    console.log(
      `Attachment is the file '${
        attachment.fileName
      }' with data: \n${new TextDecoder().decode(attachment.data)}`,
    );

    // Confirm that the support case should be resolved.
    const isResolved = await resolveCase(caseId);
    if (isResolved) {
      // List the resolved cases and include the one previously created.
      // Resolved cases can take a while to appear.
      console.log(
        "\nWaiting for case status to be marked as resolved. This can take some
 time.",
      );
      const resolvedCases = await retry(
        { intervalInMs: 20000, maxRetries: 15 },
        () => getTodaysResolvedCases(caseId),
      );
      console.log("Resolved cases:");
      console.log(resolvedCases.map((c) => c.caseId).join("\n"));
    }
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases
  - DescribeCommunications
  - DescribeServices

- DescribeSeverityLevels
- ResolveCase

Actions

`AddAttachmentsToSet`

L'exemple de code suivant montre comment utiliser`AddAttachmentsToSet`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { AddAttachmentsToSetCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new attachment set or add attachments to an existing set.
    // Provide an 'attachmentSetId' value to add attachments to an existing set.
    // Use AddCommunicationToCase or CreateCase to associate an attachment set with
 a support case.
    const response = await client.send(
      new AddAttachmentsToSetCommand({
        // You can add up to three attachments per set. The size limit is 5 MB per
 attachment.
        attachments: [
          {
            fileName: "example.txt",
            data: new TextEncoder().encode("some example text"),
          },
        ],
      }),
    );
    // Use this ID in AddCommunicationToCase or CreateCase.
    console.log(response.attachmentSetId);
```

```
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous AddAttachmentsToSetà la section Référence des AWS SDK pour JavaScript API.

`AddCommunicationToCase`

L'exemple de code suivant montre comment utiliser`AddCommunicationToCase`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { AddCommunicationToCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  let attachmentSetId;

  try {
    // Add a communication to a case.
    const response = await client.send(
      new AddCommunicationToCaseCommand({
        communicationBody: "Adding an attachment.",
        // Set value to an existing support case id.
        caseId: "CASE_ID",
        // Optional. Set value to an existing attachment set id to add attachments
 to the case.
        attachmentSetId,
      }),
    );
```

```
    console.log(response);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous AddCommunicationToCaseà la section Référence des AWS SDK pour JavaScript API.

`CreateCase`

L'exemple de code suivant montre comment utiliser`CreateCase`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new case and log the case id.
    // Important: This creates a real support case in your account.
    const response = await client.send(
      new CreateCaseCommand({
        // The subject line of the case.
        subject: "IGNORE: Test case",
        // Use DescribeServices to find available service codes for each service.
        serviceCode: "service-quicksight-end-user",
        // Use DescribeSecurityLevels to find available severity codes for your
 support plan.
        severityCode: "low",
        // Use DescribeServices to find available category codes for each service.
```

```
        categoryCode: "end-user-support",
        // The main description of the support case.
        communicationBody: "This is a test. Please ignore.",
      }),
    );
    console.log(response.caseId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateCaseà la section Référence des AWS SDK pour JavaScript API.

`DescribeAttachment`

L'exemple de code suivant montre comment utiliser`DescribeAttachment`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeAttachmentCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the metadata and content of an attachment.
    const response = await client.send(
      new DescribeAttachmentCommand({
        // Set value to an existing attachment id.
        // Use DescribeCommunications or DescribeCases to find an attachment id.
        attachmentId: "ATTACHMENT_ID",
      }),
```

```
    );
    console.log(response.attachment?.fileName);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeAttachmentà la section Référence des AWS SDK pour JavaScript API.

`DescribeCases`

L'exemple de code suivant montre comment utiliser`DescribeCases`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeCasesCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all of the unresolved cases in your account.
    // Filter or expand results by providing parameters to the DescribeCasesCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecasescommandinput.html
    const response = await client.send(new DescribeCasesCommand({}));
    const caseIds = response.cases.map((supportCase) => supportCase.caseId);
    console.log(caseIds);
    return response;
```

```
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeCasesà la section Référence des AWS SDK pour JavaScript API.

`DescribeCommunications`

L'exemple de code suivant montre comment utiliser`DescribeCommunications`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeCommunicationsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all communications for the support case.
    // Filter results by providing parameters to the DescribeCommunicationsCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecommunicationscommandinput.html
    const response = await client.send(
      new DescribeCommunicationsCommand({
        // Set value to an existing case id.
        caseId: "CASE_ID",
      }),
    );
    const text = response.communications.map((item) => item.body).join("\n");
```

```
    console.log(text);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeCommunicationsà la section Référence des AWS SDK pour JavaScript API.

`DescribeSeverityLevels`

L'exemple de code suivant montre comment utiliser`DescribeSeverityLevels`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DescribeSeverityLevelsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the list of severity levels.
    // The available values depend on the support plan for the account.
    const response = await client.send(new DescribeSeverityLevelsCommand({}));
    console.log(response.severityLevels);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeSeverityLevelsà la section Référence des AWS SDK pour JavaScript API.

`ResolveCase`

L'exemple de code suivant montre comment utiliser`ResolveCase`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { ResolveCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

const main = async () => {
  try {
    const response = await client.send(
      new ResolveCaseCommand({
        caseId: "CASE_ID",
      }),
    );

    console.log(response.finalCaseStatus);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ResolveCaseà la section Référence des AWS SDK pour JavaScript API.

Exemples de Systems Manager utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la AWS SDK pour JavaScript version (v3) avec Systems Manager.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Mise en route

Bonjour Systems Manager

L'exemple de code suivant montre comment commencer à utiliser Systems Manager.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { paginateListDocuments, SSMClient } from "@aws-sdk/client-ssm";
```

```
// Call ListDocuments and display the result.
export const main = async () => {
  const client = new SSMClient();
  const listDocumentsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListDocuments({ client }, { MaxResults: 5 });
    for await (const page of paginator) {
      listDocumentsPaginated.push(...page.DocumentIdentifiers);
    }
  } catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
  }

  for (const { Name, DocumentFormat, CreatedDate } of listDocumentsPaginated) {
    console.log(`${Name} - ${DocumentFormat} - ${CreatedDate}`);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Pour plus de détails sur l'API, reportez-vous ListDocumentsà la section Référence des AWS SDK pour JavaScript API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- créer une fenêtre de maintenance ;
- modifier le calendrier de la fenêtre de maintenance ;
- créer un document ;

- Envoyez une commande à une EC2 instance spécifiée.
- Créez un OpsItem.
- Mettez à jour et corrigez le OpsItem.
- Supprimez la fenêtre de maintenance OpsItem et le document.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { fileURLToPath } from "node:url";
import {
  CreateDocumentCommand,
  CreateMaintenanceWindowCommand,
  CreateOpsItemCommand,
  DeleteDocumentCommand,
  DeleteMaintenanceWindowCommand,
  DeleteOpsItemCommand,
  DescribeOpsItemsCommand,
  DocumentAlreadyExists,
  OpsItemStatus,
  waitUntilCommandExecuted,
  CancelCommandCommand,
  paginateListCommandInvocations,
  SendCommandCommand,
  UpdateMaintenanceWindowCommand,
  UpdateOpsItemCommand,
  SSMClient,
} from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";
```

```
/**
 * @typedef {{
 *   ssmClient: import('@aws-sdk/client-ssm').SSMClient,
 *   documentName?: string
 *   maintenanceWindow?: string
 *   winId?: int
 *   ec2InstanceId?: string
 *   requestedDateTime?: Date
 *   opsItemId?: string
 *   askToDeleteResources?: boolean
 * }} State
 */

const defaultMaintenanceWindow = "ssm-maintenance-window";
const defaultDocumentName = "ssmdocument";
// The timeout duration is highly dependent on the specific setup and environment
 necessary. This example handles only the most common error cases, and uses a much
 shorter duration than most productions systems would use.
const COMMAND_TIMEOUT_DURATION_SECONDS = 30; // 30 seconds

const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `Welcome to the AWS Systems Manager SDK Getting Started scenario.
    This program demonstrates how to interact with Systems Manager using the AWS SDK
 for JavaScript V3.
    Systems Manager is the operations hub for your AWS applications and resources
 and a secure end-to-end management solution.
    The program's primary functions include creating a maintenance window, creating
 a document, sending a command to a document,
    listing documents, listing commands, creating an OpsItem, modifying an OpsItem,
 and deleting Systems Manager resources.
    Upon completion of the program, all AWS resources are cleaned up.
    Let's get started...`,
  { header: true },
);

const createMaintenanceWindow = new ScenarioOutput(
  "createMaintenanceWindow",
  "Step 1: Create a Systems Manager maintenance window.",
);
```

```
const getMaintenanceWindow = new ScenarioInput(
  "maintenanceWindow",
  "Please enter the maintenance window name:",
  { type: "input", default: defaultMaintenanceWindow },
);

export const sdkCreateMaintenanceWindow = new ScenarioAction(
  "sdkCreateMaintenanceWindow",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new CreateMaintenanceWindowCommand({
          Name: state.maintenanceWindow,
          Schedule: "cron(0 10 ? * MON-FRI *)", //The schedule of the maintenance
 window in the form of a cron or rate expression.
          Duration: 2, //The duration of the maintenance window in hours.
          Cutoff: 1, //The number of hours before the end of the maintenance window
 that Amazon Web Services Systems Manager stops scheduling new tasks for execution.
          AllowUnassociatedTargets: true, //Allow the maintenance window to run on
 managed nodes, even if you haven't registered those nodes as targets.
        }),
      );
      state.winId = response.WindowId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while creating the maintenance window. Please fix the
 error and try again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const modifyMaintenanceWindow = new ScenarioOutput(
  "modifyMaintenanceWindow",
  "Modify the maintenance window by changing the schedule.",
);

const sdkModifyMaintenanceWindow = new ScenarioAction(
  "sdkModifyMaintenanceWindow",
  async (/** @type {State} */ state) => {
    try {
```

```
      await state.ssmClient.send(
        new UpdateMaintenanceWindowCommand({
          WindowId: state.winId,
          Schedule: "cron(0 0 ? * MON *)",
        }),
      );
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while modifying the maintenance window. Please fix the
 error and try again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const createSystemsManagerActions = new ScenarioOutput(
  "createSystemsManagerActions",
  "Create a document that defines the actions that Systems Manager performs on your
 EC2 instance.",
);

const getDocumentName = new ScenarioInput(
  "documentName",
  "Please enter the document: ",
  { type: "input", default: defaultDocumentName },
);

const sdkCreateSSMDoc = new ScenarioAction(
  "sdkCreateSSMDoc",
  async (/** @type {State} */ state) => {
    const contentData = `{
                "schemaVersion": "2.2",
                "description": "Run a simple shell command",
                "mainSteps": [
                    {
                        "action": "aws:runShellScript",
                        "name": "runEchoCommand",
                        "inputs": {
                          "runCommand": [
                            "echo 'Hello, world!'"
                          ]
                        }
```

```
                    }
                ]
            }`;
    try {
      await state.ssmClient.send(
        new CreateDocumentCommand({
          Content: contentData,
          Name: state.documentName,
          DocumentType: "Command",
        }),
      );
    } catch (caught) {
      console.log(`Exception type: (${typeof caught})`);
      if (caught instanceof DocumentAlreadyExists) {
        console.log("Document already exists. Continuing...\n");
      } else {
        console.error(caught.message);
        console.log(
          `An error occurred while creating the document. Please fix the error and
 try again. Error message: ${caught.message}`,
        );
        throw caught;
      }
    }
  },
);

const ec2HelloWorld = new ScenarioOutput(
  "ec2HelloWorld",
  `Now you have the option of running a command on an EC2 instance that echoes
 'Hello, world!'. In order to run this command, you must provide the instance ID
 of a Linux EC2 instance. If you do not already have a running Linux EC2 instance
 in your account, you can create one using the AWS console. For information about
 creating an EC2 instance, see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/
ec2-launch-instance-wizard.html.`,
);

const enterIdOrSkipEC2HelloWorld = new ScenarioInput(
  "enterIdOrSkipEC2HelloWorld",
  "Enter your EC2 InstanceId or press enter to skip this step: ",
  { type: "input", default: "" },
);

const sdkEC2HelloWorld = new ScenarioAction(
```

```
  "sdkEC2HelloWorld",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new SendCommandCommand({
          DocumentName: state.documentName,
          InstanceIds: [state.ec2InstanceId],
          TimeoutSeconds: COMMAND_TIMEOUT_DURATION_SECONDS,
        }),
      );
      state.CommandId = response.Command.CommandId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while sending the command. Please fix the error and try
 again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      state.enterIdOrSkipEC2HelloWorld === "",
  },
);

const sdkGetCommandTime = new ScenarioAction(
  "sdkGetCommandTime",
  async (/** @type {State} */ state) => {
    const listInvocationsPaginated = [];
    console.log(
      "Let's get the time when the specific command was sent to the specific managed
 node.",
    );

    console.log(
      `First, we'll wait for the command to finish executing. This may take up to
 ${COMMAND_TIMEOUT_DURATION_SECONDS} seconds.`,
    );
    const commandExecutedResult = waitUntilCommandExecuted(
      { client: state.ssmClient },
      {
        CommandId: state.CommandId,
        InstanceId: state.ec2InstanceId,
```

```
      },
    );
    // This is necessary because the TimeoutSeconds of SendCommandCommand is only
for the delivery, not execution.
    try {
      await new Promise((_, reject) =>
        setTimeout(
          reject,
          COMMAND_TIMEOUT_DURATION_SECONDS * 1000,
          new Error("Command Timed Out"),
        ),
      );
    } catch (caught) {
      if (caught.message === "Command Timed Out") {
        commandExecutedResult.state = "TIMED_OUT";
      } else {
        throw caught;
      }
    }

    if (commandExecutedResult.state !== "SUCCESS") {
      console.log(
        `The command with id: ${state.CommandId} did not execute in the allotted
time. Canceling command.`,
      );
      state.ssmClient.send(
        new CancelCommandCommand({
          CommandId: state.CommandId,
        }),
      );
      state.enterIdOrSkipEC2HelloWorld === "";
      return;
    }

    for await (const page of paginateListCommandInvocations(
      { client: state.ssmClient },
      { CommandId: state.CommandId },
    )) {
      listInvocationsPaginated.push(...page.CommandInvocations);
    }
    /**
     * @type {import('@aws-sdk/client-ssm').CommandInvocation}
     */
```

```
    const commandInvocation = listInvocationsPaginated.shift(); // Because the call
 was made with CommandId, there's only one result, so shift it off.
    state.requestedDateTime = commandInvocation.RequestedDateTime;

    console.log(
      `The command invocation happened at: ${state.requestedDateTime}.`,
    );
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      state.enterIdOrSkipEC2HelloWorld === "",
  },
);

const createSSMOpsItem = new ScenarioOutput(
  "createSSMOpsItem",
  `Now we will create a Systems Manager OpsItem. An OpsItem is a feature provided by
 the Systems Manager service. It is a type of operational data item that allows you
 to manage and track various operational issues, events, or tasks within your AWS
 environment.
You can create OpsItems to track and manage operational issues as they arise. For
 example, you could create an OpsItem whenever your application detects a critical
 error or an anomaly in your infrastructure.`,
);

const sdkCreateSSMOpsItem = new ScenarioAction(
  "sdkCreateSSMOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new CreateOpsItemCommand({
          Description: "Created by the System Manager Javascript API",
          Title: "Disk Space Alert",
          Source: "EC2",
          Category: "Performance",
          Severity: "2",
        }),
      );
      state.opsItemId = response.OpsItemId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while creating the ops item. Please fix the error and try
 again. Error message: ${caught.message}`,
```

```
      );
      throw caught;
    }
  },
);

const updateOpsItem = new ScenarioOutput(
  "updateOpsItem",
  (/** @type {State} */ state) =>
    `Now we will update the OpsItem: ${state.opsItemId}`,
);

const sdkUpdateOpsItem = new ScenarioAction(
  "sdkUpdateOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const _response = await state.ssmClient.send(
        new UpdateOpsItemCommand({
          OpsItemId: state.opsItemId,
          Description: `An update to ${state.opsItemId}`,
        }),
      );
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while updating the ops item. Please fix the error and try
 again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const getOpsItemStatus = new ScenarioOutput(
  "getOpsItemStatus",
  (/** @type {State} */ state) =>
    `Now we will get the status of the OpsItem: ${state.opsItemId}`,
);

const sdkOpsItemStatus = new ScenarioAction(
  "sdkGetOpsItemStatus",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
```

```
        new DescribeOpsItemsCommand({
          OpsItemId: state.opsItemId,
        }),
      );
      state.opsItemStatus = response.OpsItemStatus;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while describing the ops item. Please fix the error and
 try again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const resolveOpsItem = new ScenarioOutput(
  "resolveOpsItem",
  (/** @type {State} */ state) =>
    `Now we will resolve the OpsItem: ${state.opsItemId}`,
);

const sdkResolveOpsItem = new ScenarioAction(
  "sdkResolveOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const _response = await state.ssmClient.send(
        new UpdateOpsItemCommand({
          OpsItemId: state.opsItemId,
          Status: OpsItemStatus.RESOLVED,
        }),
      );
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while updating the ops item. Please fix the error and try
 again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const askToDeleteResources = new ScenarioInput(
```

```
  "askToDeleteResources",
  "Would you like to delete the Systems Manager resources created during this
 example run?",
  { type: "confirm" },
);

const confirmDeleteChoice = new ScenarioOutput(
  "confirmDeleteChoice",
  (/** @type {State} */ state) => {
    if (state.askToDeleteResources) {
      return "You chose to delete the resources.";
    }
    return "The Systems Manager resources will not be deleted. Please delete them
 manually to avoid charges.";
  },
);

export const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    try {
      await state.ssmClient.send(
        new DeleteOpsItemCommand({
          OpsItemId: state.opsItemId,
        }),
      );
      console.log(`The ops item: ${state.opsItemId} was successfully deleted.`);
    } catch (caught) {
      console.log(
        `There was a problem deleting the ops item: ${state.opsItemId}. Please
 delete it manually. Error: ${caught.message}`,
      );
    }

    try {
      await state.ssmClient.send(
        new DeleteMaintenanceWindowCommand({
          Name: state.maintenanceWindow,
          WindowId: state.winId,
        }),
      );
      console.log(
        `The maintenance window: ${state.maintenanceWindow} was successfully
 deleted.`,
```

```
      );
    } catch (caught) {
      console.log(
        `There was a problem deleting the maintenance window: ${state.opsItemId}.
 Please delete it manually. Error: ${caught.message}`,
      );
    }

    try {
      await state.ssmClient.send(
        new DeleteDocumentCommand({
          Name: state.documentName,
        }),
      );
      console.log(
        `The document: ${state.documentName} was successfully deleted.`,
      );
    } catch (caught) {
      console.log(
        `There was a problem deleting the document: ${state.documentName}. Please
 delete it manually. Error: ${caught.message}`,
      );
    }
  },
  { skipWhen: (/** @type {{}} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the Systems Manager Basics scenario for the AWS Javascript SDK v3.
 Thank you!",
);

const myScenario = new Scenario(
  "SSM Basics",
  [
    greet,
    pressEnter,
    createMaintenanceWindow,
    getMaintenanceWindow,
    sdkCreateMaintenanceWindow,
    modifyMaintenanceWindow,
    pressEnter,
    sdkModifyMaintenanceWindow,
```

```
    createSystemsManagerActions,
    getDocumentName,
    sdkCreateSSMDoc,
    ec2HelloWorld,
    enterIdOrSkipEC2HelloWorld,
    sdkEC2HelloWorld,
    sdkGetCommandTime,
    pressEnter,
    createSSMOpsItem,
    pressEnter,
    sdkCreateSSMOpsItem,
    updateOpsItem,
    pressEnter,
    sdkUpdateOpsItem,
    getOpsItemStatus,
    pressEnter,
    sdkOpsItemStatus,
    resolveOpsItem,
    pressEnter,
    sdkResolveOpsItem,
    askToDeleteResources,
    confirmDeleteChoice,
    sdkDeleteResources,
    goodbye,
  ],
  { ssmClient: new SSMClient({}) },
);

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

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
```

```
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour JavaScript .
  - CreateDocument
  - CreateMaintenanceWindow
  - CreateOpsItem
  - DeleteMaintenanceWindow
  - ListCommandInvocations
  - SendCommand
  - UpdateOpsItem

Actions

`CreateDocument`

L'exemple de code suivant montre comment utiliser`CreateDocument`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateDocumentCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ content, name, documentType }) => {
  const client = new SSMClient({});
  try {
    const { documentDescription } = await client.send(
```

```
      new CreateDocumentCommand({
        Content: content, // The content for the new SSM document. The content must
 not exceed 64KB.
        Name: name,
        DocumentType: documentType, // Document format type can be JSON, YAML, or
 TEXT. The default format is JSON.
      }),
    );
    console.log("Document created successfully.");
    return { DocumentDescription: documentDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DocumentAlreadyExists") {
      console.warn(`${caught.message}. Did you provide a new document name?`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateDocumentà la section Référence des AWS SDK pour JavaScript API.

`CreateMaintenanceWindow`

L'exemple de code suivant montre comment utiliser`CreateMaintenanceWindow`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM maintenance window.
```

```
 * @param {{ name: string, allowUnassociatedTargets: boolean, duration: number,
 cutoff: number, schedule: string, description?: string }}
 */
export const main = async ({
  name,
  allowUnassociatedTargets, // Allow the maintenance window to run on managed nodes,
 even if you haven't registered those nodes as targets.
  duration, // The duration of the maintenance window in hours.
  cutoff, // The number of hours before the end of the maintenance window that
 Amazon Web Services Systems Manager stops scheduling new tasks for execution.
  schedule, // The schedule of the maintenance window in the form of a cron or rate
 expression.
  description = undefined,
}) => {
  const client = new SSMClient({});

  try {
    const { windowId } = await client.send(
      new CreateMaintenanceWindowCommand({
        Name: name,
        Description: description,
        AllowUnassociatedTargets: allowUnassociatedTargets, // Allow the maintenance
 window to run on managed nodes, even if you haven't registered those nodes as
 targets.
        Duration: duration, // The duration of the maintenance window in hours.
        Cutoff: cutoff, // The number of hours before the end of the maintenance
 window that Amazon Web Services Systems Manager stops scheduling new tasks for
 execution.
        Schedule: schedule, // The schedule of the maintenance window in the form of
 a cron or rate expression.
      }),
    );
    console.log(`Maintenance window created with Id: ${windowId}`);
    return { WindowId: windowId };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateMaintenanceWindowà la section Référence des AWS SDK pour JavaScript API.

`CreateOpsItem`

L'exemple de code suivant montre comment utiliser`CreateOpsItem`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { CreateOpsItemCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM OpsItem.
 * @param {{ title: string, source: string, category?: string, severity?: string }}
 */
export const main = async ({
  title,
  source,
  category = undefined,
  severity = undefined,
}) => {
  const client = new SSMClient({});
  try {
    const { opsItemArn, opsItemId } = await client.send(
      new CreateOpsItemCommand({
        Title: title,
        Source: source, // The origin of the OpsItem, such as Amazon EC2 or Systems
 Manager.
        Category: category,
        Severity: severity,
      }),
    );
    console.log(`Ops item created with id: ${opsItemId}`);
    return { OpsItemArn: opsItemArn, OpsItemId: opsItemId };
  } catch (caught) {
```

```
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateOpsItemà la section Référence des AWS SDK pour JavaScript API.

`DeleteDocument`

L'exemple de code suivant montre comment utiliser`DeleteDocument`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteDocumentCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Delete an SSM document.
 * @param {{ documentName: string }}
 */
export const main = async ({ documentName }) => {
  const client = new SSMClient({});
  try {
    await client.send(new DeleteDocumentCommand({ Name: documentName }));
    console.log(`Document '${documentName}' deleted.`);
    return { Deleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    } else {
```

```
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteDocumentà la section Référence des AWS SDK pour JavaScript API.

`DeleteMaintenanceWindow`

L'exemple de code suivant montre comment utiliser`DeleteMaintenanceWindow`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { DeleteMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Delete an SSM maintenance window.
 * @param {{ windowId: string }}
 */
export const main = async ({ windowId }) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new DeleteMaintenanceWindowCommand({ WindowId: windowId }),
    );
    console.log(`Maintenance window '${windowId}' deleted.`);
    return { Deleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    } else {
      throw caught;
```

```
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteMaintenanceWindowà la section Référence des AWS SDK pour JavaScript API.

`DescribeOpsItems`

L'exemple de code suivant montre comment utiliser`DescribeOpsItems`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import {
  OpsItemFilterOperator,
  OpsItemFilterKey,
  paginateDescribeOpsItems,
  SSMClient,
} from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Describe SSM OpsItems.
 * @param {{ opsItemId: string }}
 */
export const main = async ({ opsItemId }) => {
  const client = new SSMClient({});
  try {
    const describeOpsItemsPaginated = [];
    for await (const page of paginateDescribeOpsItems(
      { client },
      {
        OpsItemFilters: {
          Key: OpsItemFilterKey.OPSITEM_ID,
```

```
          Operator: OpsItemFilterOperator.EQUAL,
          Values: opsItemId,
        },
      },
    )) {
      describeOpsItemsPaginated.push(...page.OpsItemSummaries);
    }
    console.log("Here are the ops items:");
    console.log(describeOpsItemsPaginated);
    return { OpsItemSummaries: describeOpsItemsPaginated };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    }
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeOpsItemsà la section Référence des AWS SDK pour JavaScript API.

`ListCommandInvocations`

L'exemple de code suivant montre comment utiliser`ListCommandInvocations`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { paginateListCommandInvocations, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * List SSM command invocations on an instance.
 * @param {{ instanceId: string }}
 */
```

```
export const main = async ({ instanceId }) => {
  const client = new SSMClient({});
  try {
    const listCommandInvocationsPaginated = [];
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListCommandInvocations(
      { client },
      {
        InstanceId: instanceId,
      },
    );
    for await (const page of paginator) {
      listCommandInvocationsPaginated.push(...page.CommandInvocations);
    }
    console.log("Here is the list of command invocations:");
    console.log(listCommandInvocationsPaginated);
    return { CommandInvocations: listCommandInvocationsPaginated };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
      console.warn(`${caught.message}. Did you provide a valid instance ID?`);
    }
    throw caught;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ListCommandInvocationsà la section Référence des AWS SDK pour JavaScript API.

`SendCommand`

L'exemple de code suivant montre comment utiliser`SendCommand`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { SendCommandCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Send an SSM command to a managed node.
 * @param {{ documentName: string }}
 */
export const main = async ({ documentName }) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new SendCommandCommand({
        DocumentName: documentName,
      }),
    );
    console.log("Command sent successfully.");
    return { Success: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
      console.warn(`${caught.message}. Did you provide a valid document name?`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous SendCommandà la section Référence des AWS SDK pour JavaScript API.

`UpdateMaintenanceWindow`

L'exemple de code suivant montre comment utiliser`UpdateMaintenanceWindow`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { UpdateMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Update an SSM maintenance window.
 * @param {{ windowId: string, allowUnassociatedTargets?: boolean, duration?:
 number, enabled?: boolean, name?: string, schedule?: string }}
 */
export const main = async ({
  windowId,
  allowUnassociatedTargets = undefined, //Allow the maintenance window to run on
 managed nodes, even if you haven't registered those nodes as targets.
  duration = undefined, //The duration of the maintenance window in hours.
  enabled = undefined,
  name = undefined,
  schedule = undefined, //The schedule of the maintenance window in the form of a
 cron or rate expression.
}) => {
  const client = new SSMClient({});
  try {
    const { opsItemArn, opsItemId } = await client.send(
      new UpdateMaintenanceWindowCommand({
        WindowId: windowId,
        AllowUnassociatedTargets: allowUnassociatedTargets,
        Duration: duration,
        Enabled: enabled,
        Name: name,
        Schedule: schedule,
      }),
    );
    console.log("Maintenance window updated.");
    return { OpsItemArn: opsItemArn, OpsItemId: opsItemId };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
      console.warn(`${caught.message}. Are these values correct?`);
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous UpdateMaintenanceWindowà la section Référence des AWS SDK pour JavaScript API.

`UpdateOpsItem`

L'exemple de code suivant montre comment utiliser`UpdateOpsItem`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { UpdateOpsItemCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Update an SSM OpsItem.
 * @param {{ opsItemId: string, status?: OpsItemStatus }}
 */
export const main = async ({
  opsItemId,
  status = undefined, // The OpsItem status. Status can be Open, In Progress, or
 Resolved
}) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new UpdateOpsItemCommand({
        OpsItemId: opsItemId,
        Status: status,
      }),
    );
    console.log("Ops item updated.");
    return { Success: true };
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "OpsItemLimitExceededException"
    ) {
```

```
      console.warn(
        `Couldn't create ops item because you have exceeded your open OpsItem limit.
 ${caught.message}.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Pour plus de détails sur l'API, reportez-vous UpdateOpsItemà la section Référence des AWS SDK pour JavaScript API.

Exemples d'Amazon Textract utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Textract.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Scénarios

Scénarios

Créer une application Amazon Textract Explorer

L'exemple de code suivant montre comment explorer la sortie Amazon Textract via une application interactive.

SDK pour JavaScript (v3)

> Montre comment utiliser le AWS SDK pour JavaScript pour créer une application React qui utilise Amazon Textract pour extraire des données d'une image de document et les afficher sur une

page Web interactive. Cet exemple s'exécute dans un navigateur Web et nécessite une identité Amazon Cognito authentifiée pour les informations d'identification. Il utilise Amazon Simple Storage Service (Amazon S3) pour le stockage et, pour les notifications, il interroge une file d'attente Amazon Simple Queue Service (Amazon SQS) abonnée à une rubrique Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Créez une application pour analyser les commentaires des clients

L'exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour JavaScript (v3)

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub. Les extraits suivants montrent comment le AWS SDK pour JavaScript est utilisé dans les fonctions Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";
```

```
/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
```

```
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});
```

```
  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Exemples d'Amazon Transcribe utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Transcribe.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

- Scénarios

Actions

`DeleteMedicalTranscriptionJob`

L'exemple de code suivant montre comment utiliser`DeleteMedicalTranscriptionJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Supprimez une tâche de transcription médicale.

```
// 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();
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteMedicalTranscriptionJobà la section Référence des AWS SDK pour JavaScript API.

`DeleteTranscriptionJob`

L'exemple de code suivant montre comment utiliser`DeleteTranscriptionJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez une tâche de transcription.

```
// 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();
```

Créez le client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteTranscriptionJobà la section Référence des AWS SDK pour JavaScript API.

`ListMedicalTranscriptionJobs`

L'exemple de code suivant montre comment utiliser`ListMedicalTranscriptionJobs`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Répertoriez les tâches de transcription médicale.

```
// 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();
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListMedicalTranscriptionJobsà la section Référence des AWS SDK pour JavaScript API.

`ListTranscriptionJobs`

L'exemple de code suivant montre comment utiliser`ListTranscriptionJobs`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les tâches de transcription.

```
// 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();
```

Créez le client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListTranscriptionJobsà la section Référence des AWS SDK pour JavaScript API.

`StartMedicalTranscriptionJob`

L'exemple de code suivant montre comment utiliser`StartMedicalTranscriptionJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez le client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Démarrez une tâche de transcription médicale.

```
// 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();
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous StartMedicalTranscriptionJobà la section Référence des AWS SDK pour JavaScript API.

`StartTranscriptionJob`

L'exemple de code suivant montre comment utiliser`StartTranscriptionJob`.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Démarrez une tâche de transcription.

```
// 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();
```

Créez le client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Pour plus d'informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous StartTranscriptionJobà la section Référence des AWS SDK pour JavaScript API.

Scénarios

Créer une application de streaming Amazon Transcribe

L'exemple de code suivant montre comment créer une application qui enregistre, transcrit et traduit de l'audio en direct en temps réel, et envoie les résultats par e-mail.

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Transcribe afin de créer une application qui enregistre, transcrit et traduit de l'audio en direct en temps réel, et envoie les résultats par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Exemples d'Amazon Translate utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la version AWS SDK pour JavaScript (v3) avec Amazon Translate.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

Scénarios

Créer une application de streaming Amazon Transcribe

L'exemple de code suivant montre comment créer une application qui enregistre, transcrit et traduit de l'audio en direct en temps réel, et envoie les résultats par e-mail.

SDK pour JavaScript (v3)

> Montre comment utiliser Amazon Transcribe afin de créer une application qui enregistre, transcrit et traduit de l'audio en direct en temps réel, et envoie les résultats par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).
>
> Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.
>
> Les services utilisés dans cet exemple
>
> - Amazon Comprehend
> - Amazon SES
> - Amazon Transcribe
> - Amazon Translate

Création d’un chatbot Amazon Lex

L’exemple de code suivant montre comment créer un chatbot pour interagir avec les visiteurs de votre site Web.

SDK pour JavaScript (v3)

Indique comment utiliser l’API Amazon Lex pour créer un chatbot dans une application Web afin d’interagir avec les visiteurs de votre site Web.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet de création d'un chatbot Amazon Lex dans le guide du AWS SDK pour JavaScript développeur.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Créez une application pour analyser les commentaires des clients

L’exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d’origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour JavaScript (v3)

Cet exemple d’application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d’un hôtel fictif situé à New York. L’hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l’application via un client Web. Après avoir chargé l’image d’une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l’image à l’aide d’Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l’aide d’Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub. Les extraits suivants montrent comment le AWS SDK pour JavaScript est utilisé dans les fonctions Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
```

```
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
```

```
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Sécurité de ce AWS produit ou service

Chez Amazon Web Services (AWS), la sécurité dans le cloud est la priorité principale. En tant que client AWS , vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des organisations les plus pointilleuses sur la sécurité. La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cela comme la sécurité du cloud et la sécurité dans le cloud.

Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute tous les services proposés dans le AWS cloud et de vous fournir des services que vous pouvez utiliser en toute sécurité. Notre responsabilité en matière de sécurité est notre priorité absolue AWS, et l'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des programmes de AWS conformité.

Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez et par d'autres facteurs, notamment la sensibilité de vos données, les exigences de votre organisation et les lois et réglementations applicables.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Rubriques

- Protection des données dans ce AWS produit ou service
- Gestion de l’identité et des accès
- Validation de conformité pour ce AWS produit ou service
- Résilience pour ce AWS produit ou service
- Sécurité de l'infrastructure pour ce AWS produit ou service
- Appliquer une version minimale de TLS

Protection des données dans ce AWS produit ou service

Le modèle de responsabilité AWS partagée de s'applique à la protection des données dans ce AWS produit ou service. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur

cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez Questions fréquentes (FAQ) sur la confidentialité des données. Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog Modèle de responsabilité partagée d'AWS et RGPD (Règlement général sur la protection des données) sur le Blog de sécuritéAWS .

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou Gestion des identités et des accès AWS (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- SSL/TLS À utiliser pour communiquer avec AWS les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section Utilisation des CloudTrail sentiers dans le guide de AWS CloudTrail l'utilisateur.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés par la norme FIPS 140-3 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS disponibles, consultez Norme FIPS (Federal Information Processing Standard) 140-3.

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Nom. Cela inclut lorsque vous travaillez avec ce AWS produit ou service ou un autre à Services AWS l'aide de la console, de l'API ou AWS SDKs. AWS CLI Toutes les données que vous entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Gestion de l’identité et des accès

Gestion des identités et des accès AWS (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser AWS les ressources. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction du travail que vous effectuez. AWS

Utilisateur du service : si vous avez l' Services AWS habitude de faire votre travail, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Au fur et à mesure que vous utilisez de nouvelles AWS fonctionnalités pour effectuer votre travail, vous aurez peut-être besoin d'autorisations supplémentaires. Si vous comprenez bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne pouvez pas accéder à une fonctionnalité dans AWS, consultez Résolution des problèmes AWS d'identité et d'accès le guide de l'utilisateur du Service AWS que vous utilisez.

Administrateur du service — Si vous êtes responsable des AWS ressources de votre entreprise, vous avez probablement un accès complet à AWS. C'est à vous de déterminer les AWS fonctionnalités et les ressources auxquelles les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d’IAM. Pour en savoir plus sur la façon dont votre entreprise peut utiliser IAM avec AWS, consultez le guide de l'utilisateur Service AWS que vous utilisez.

Administrateur IAM – Si vous êtes un administrateur IAM, vous souhaiterez peut-être en savoir plus sur la façon d’écrire des politiques pour gérer l’accès à AWS. Pour consulter des exemples

de politiques AWS basées sur l'identité que vous pouvez utiliser dans IAM, consultez le guide de l'utilisateur Service AWS que vous utilisez.

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez Connexion à votre Compte AWS dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez Signature AWS Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez Tâches qui requièrent les informations d'identification de l'utilisateur racine dans le Guide de l'utilisateur IAM.

Identité fédérée

Il est recommandé d'obliger les utilisateurs humains à utiliser la fédération avec un fournisseur d'identité pour accéder à Services AWS l'aide d'informations d'identification temporaires.

Une identité fédérée est un utilisateur provenant de l'annuaire de votre entreprise, de votre fournisseur d'identité Web ou Directory Service qui y accède à Services AWS l'aide d'informations d'identification provenant d'une source d'identité. Les identités fédérées assument des rôles qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Pour plus d'informations, consultez Qu'est-ce que IAM Identity Center ? dans le Guide de l'utilisateur AWS IAM Identity Center .

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires dans le guide de l'utilisateur IAM.

Les groupes IAM spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez Cas d'utilisation pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en passant d'un rôle d'utilisateur à un rôle IAM (console) ou en appelant une opération d' AWS API AWS CLI ou d'API. Pour plus d'informations, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès entre comptes, les accès entre services et pour les applications exécutées sur Amazon. EC2 Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Politiques basées sur l’identité

Les stratégies basées sur l’identité sont des documents de stratégie d’autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l’identité, consultez Définition d’autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l’utilisateur IAM.

Les politiques basées sur l’identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l’utilisateur IAM.

Politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Les exemples incluent les politiques de confiance de rôle IAM et les stratégies de compartiment Amazon S3. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l’accès à une ressource spécifique. Vous devez spécifier un principal dans une politique basée sur les ressources.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Listes de contrôle d'accès (ACLs)

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la présentation de la liste de contrôle d'accès (ACL) dans le guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez Politiques de contrôle de service dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir Politiques de contrôle des ressources (RCPs) dans le guide de AWS Organizations l'utilisateur.
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez Politiques de session dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

Comment Services AWS travailler avec IAM

Pour obtenir une vue d'ensemble du Services AWS fonctionnement de la plupart des fonctionnalités IAM, consultez les AWS services compatibles avec IAM dans le guide de l'utilisateur IAM.

Pour savoir comment utiliser un service spécifique Service AWS avec IAM, consultez la section relative à la sécurité du guide de l'utilisateur du service concerné.

Résolution des problèmes AWS d'identité et d'accès

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec AWS IAM.

Rubriques

- Je ne suis pas autorisé à effectuer une action dans AWS
- Je ne suis pas autorisé à effectuer iam : PassRole

- Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS ressources

Je ne suis pas autorisé à effectuer une action dans AWS

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit quand l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource *`my-example-widget`* fictive, mais ne dispose pas des autorisations `awes:`*`GetWidget`* fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource *`my-example-widget`* à l'aide de l'action `awes:`*`GetWidget`*.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter `iam:PassRole` l'action, vos stratégies doivent être mises à jour afin de vous permettre de transmettre un rôle à AWS.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans AWS. Toutefois, l'action nécessite que le service ait des autorisations accordées par une fonction de service. Mary n'est pas autorisée à transmettre le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS ressources

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si ces fonctionnalités sont prises AWS en charge, consultezComment Services AWS travailler avec IAM.
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Validation de conformité pour ce AWS produit ou service

Pour savoir si un programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez AWS la documentation de sécurité.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Résilience pour ce AWS produit ou service

L'infrastructure AWS mondiale est construite autour Régions AWS de zones de disponibilité.

Régions AWS fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant.

Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont davantage disponibles, tolérantes aux pannes et ont une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section Infrastructure AWS mondiale.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Sécurité de l'infrastructure pour ce AWS produit ou service

Ce AWS produit ou service utilise des services gérés et est donc protégé par la sécurité du réseau AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section Sécurité du AWS cloud. Pour concevoir votre

AWS environnement en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section Protection de l'infrastructure dans le cadre AWS bien architecturé du pilier de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder à ce AWS produit ou service via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser AWS Security Token Service (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Appliquer une version minimale de TLS

Pour renforcer la sécurité lors de la communication avec les AWS services, configurez le AWS SDK pour JavaScript pour utiliser le protocole TLS 1.2 ou version ultérieure.

Transport Layer Security (TLS) est un protocole utilisé par les navigateurs web et d'autres applications pour assurer la confidentialité et l'intégrité des données échangées sur un réseau.

> **Important**
>
> Depuis le 10 juin 2024, nous avons annoncé que le protocole TLS 1.3 est disponible sur les points de terminaison des API de AWS service dans chacune des AWS régions. La AWS SDK pour JavaScript v3 ne négocie pas la version TLS elle-même. Il utilise plutôt la version TLS déterminée par Node.js, qui est configurable via `https.Agent`. AWS recommande d'utiliser la version Active LTS actuelle de Node.js.

Vérifier et appliquer TLS dans Node.js

Lorsque vous utilisez le AWS SDK pour JavaScript with Node.js, la couche de sécurité Node.js sous-jacente est utilisée pour définir la version TLS.

Node.js 12.0.0 et versions ultérieures utilisent une version minimale d'OpenSSL 1.1.1b, qui prend en charge le protocole TLS 1.3. Node.js utilise par défaut le protocole TLS 1.3 lorsqu'il est disponible. Vous pouvez spécifier une version différente de manière explicite si nécessaire.

Vérifier la version d'OpenSSL et TLS

Pour obtenir la version d'OpenSSL utilisée par Node.js sur votre ordinateur, exécutez la commande suivante.

```
node -p process.versions
```

La version d'OpenSSL dans la liste est la version utilisée par Node.js, comme illustré dans l'exemple suivant.

```
openssl: '1.1.1b'
```

Pour obtenir la version de TLS utilisée par Node.js sur votre ordinateur, démarrez le shell Node et exécutez les commandes suivantes, dans l'ordre.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

La dernière commande génère la version TLS, comme illustré dans l'exemple suivant.

```
'TLSv1.3'
```

Node.js utilise par défaut cette version de TLS et tente de négocier une autre version de TLS si un appel échoue.

Vérification des versions TLS minimales et maximales prises en charge

Les développeurs peuvent vérifier les versions TLS minimale et maximale prises en charge dans Node.js à l'aide du script suivant :

```
import tls from "tls";
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +
 tls.DEFAULT_MAX_VERSION);
```

La dernière commande affiche les versions TLS minimale et maximale par défaut, comme indiqué dans l'exemple suivant.

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

Appliquer une version minimale de TLS

Node.js négocie une version de TLS lorsqu'un appel échoue. Vous pouvez appliquer la version TLS minimale autorisée au cours de cette négociation, soit lors de l'exécution d'un script depuis la ligne de commande, soit par requête dans votre JavaScript code.

Pour spécifier la version minimale de TLS à partir de la ligne de commande, vous devez utiliser Node.js version 11.4.0 ou ultérieure. Pour installer une version spécifique de Node.js, installez d'abord le gestionnaire de version de Node (nvm) en suivant les étapes décrites dans la section Installation et mise à jour du gestionnaire de versions de Node. Ensuite, exécutez les commandes suivantes pour installer et utiliser une version spécifique de Node.js.

```
nvm install 11
nvm use 11
```

Enforce TLS 1.2

Pour faire en sorte que TLS 1.2 soit la version minimale autorisée, spécifiez l'argument `--tls-min-v1.2` lors de l'exécution de votre script, comme indiqué dans l'exemple suivant.

```
node --tls-min-v1.2 yourScript.js
```

Pour spécifier la version TLS minimale autorisée pour une demande spécifique dans votre JavaScript code, utilisez le `minVersion` paramètre pour spécifier le protocole, comme indiqué dans l'exemple suivant.

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                minVersion: 'TLSv1.2'
            }
        )
    })
});
```

Enforce TLS 1.3

Pour confirmer que TLS 1.3 est la version minimale autorisée, spécifiez l'`--tls-min-v1.3`argument lors de l'exécution de votre script, comme indiqué dans l'exemple suivant.

```
node --tls-min-v1.3 yourScript.js
```

Pour spécifier la version TLS minimale autorisée pour une demande spécifique dans votre JavaScript code, utilisez le `minVersion` paramètre pour spécifier le protocole, comme indiqué dans l'exemple suivant.

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                minVersion: 'TLSv1.3'
            }
        )
    })
});
```

Vérifier et appliquer TLS dans un script de navigateur

Lorsque vous utilisez le SDK JavaScript dans un script de navigateur, les paramètres du navigateur contrôlent la version de TLS utilisée. La version de TLS utilisée par le navigateur ne peut pas être

découverte ou définie par script et doit être configurée par l'utilisateur. Pour vérifier et appliquer la version de TLS utilisée dans un script de navigateur, veuillez consulter les instructions de votre navigateur spécifique.

Microsoft Internet Explorer

1. Ouvrez Internet Explorer.
2. Dans la barre de menu, choisissez Outils - Options Internet - onglet Avancé.
3. Faites défiler la page jusqu'à la catégorie de sécurité, cochez manuellement la case Utiliser TLS 1.2.
4. Cliquez sur OK.
5. Fermez votre navigateur et redémarrez Internet Explorer.

Microsoft Edge

1. Dans le champ de recherche du menu Windows, tapez*`Internet options`*.
2. Sous Meilleure correspondance, cliquez sur Options Internet.
3. Dans la fenêtre Propriétés Internet, sous l'onglet Avancé, faites défiler la page jusqu'à la section Sécurité.
4. Cochez la case User TLS 1.2.
5. Cliquez sur OK.

Google Chrome

1. Ouvrez Google Chrome.
2. Cliquez sur Alt F et sélectionnez Paramètres.
3. Faites défiler l'écran vers le bas et sélectionnez Afficher les paramètres avancés... .
4. Faites défiler la page jusqu'à la section Système et cliquez sur Ouvrir les paramètres du proxy... .
5. Sélectionnez l'onglet Avancé.
6. Faites défiler la page jusqu'à la catégorie de sécurité, cochez manuellement la case Utiliser TLS 1.2.
7. Cliquez sur OK.
8. Fermez votre navigateur et redémarrez Google Chrome.

Mozilla Firefox

1. Ouvrez Firefox.
2. Dans la barre d'adresse, tapez about:config et appuyez sur Entrée.
3. Dans le champ de recherche, saisissez tls. Recherchez et double-cliquez sur l'entrée relative à security.tls.version.min.
4. Définissez la valeur entière sur 3 pour forcer le protocole TLS 1.2 à être le protocole par défaut.
5. Cliquez sur OK.
6. Fermez votre navigateur et redémarrez Mozilla Firefox.

Apple Safari

Il n'existe aucune option permettant d'activer les protocoles SSL. Si vous utilisez Safari version 7 ou supérieure, le protocole TLS 1.2 est automatiquement activé.

Récupération de la version TLS dans AWS SDK pour JavaScript les demandes v3

Vous pouvez enregistrer la version TLS utilisée dans une demande de AWS SDK à l'aide du script suivant :

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
import tls from "tls";

const client = new S3Client({ region: "us-east-1" });

const tlsSocket = new tls.TLSSocket();

client.middlewareStack.add((next, context) => async (args) => {
  console.log(`Using TLS version: ${tlsSocket.getProtocol()}`);
    return next(args);
});
```

La dernière commande affiche la version TLS utilisée, comme indiqué dans l'exemple suivant.

```
Using TLS version: TLSv1.3
```

Migrez de la version 2.x vers la version 3.x du AWS SDK pour JavaScript

La AWS SDK pour JavaScript version 3 est une réécriture majeure de la version 2. La section décrit les différences entre les deux versions et explique comment migrer de la version 2 à la version 3 du SDK pour JavaScript.

Migrez votre code vers le SDK pour JavaScript v3 à l'aide de codemod

AWS SDK pour JavaScript la version 3 (v3) est dotée d'interfaces modernisées pour les configurations client et les utilitaires, notamment les informations d'identification, le téléchargement partitionné sur Amazon S3, le client de documents DynamoDB, les serveurs, etc. Vous pouvez trouver ce qui a changé dans la v2 et les équivalents v3 pour chaque modification dans le guide de migration sur le AWS SDK pour JavaScript GitHub dépôt.

Pour tirer pleinement parti de la AWS SDK pour JavaScript v3, nous vous recommandons d'utiliser les scripts codemod décrits ci-dessous.

Utiliser codemod pour migrer le code v2 existant

La collection de scripts Codemod aws-sdk-js-codemodpermet de migrer votre application existante AWS SDK pour JavaScript (v2) pour utiliser la v3. APIs Vous pouvez exécuter la transformation comme suit.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

Par exemple, supposons que vous avez le code suivant, qui crée un client Amazon DynamoDB à partir de la version 2 et appelle une opération. `listTables`

```
// example.ts
import AWS from "aws-sdk";

const region = "us-west-2";
const client = new AWS.DynamoDB({ region });
await client.listTables({}).promise()
```

```
  .then(console.log)
  .catch(console.error);
```

Vous pouvez exécuter notre `v2-to-v3` transformation `example.ts` comme suit.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

La transformation convertira l'importation DynamoDB en version 3, créera un client v3 et appellera l'opération comme suit. `listTables`

```
// example.ts
import { DynamoDB } from "@aws-sdk/client-dynamodb";

const region = "us-west-2";
const client = new DynamoDB({ region });
await client.listTables({})
  .then(console.log)
  .catch(console.error);
```

Nous avons mis en œuvre des transformations pour les cas d'utilisation courants. Si votre code ne se transforme pas correctement, veuillez créer un rapport de bogue ou une demande de fonctionnalité avec un exemple de code d'entrée et un code de sortie observé/attendu. Si votre cas d'utilisation spécifique est déjà signalé dans un numéro existant, montrez votre soutien par un vote positif.

Nouveautés de la version 3

La version 3 du SDK pour JavaScript (v3) contient les nouvelles fonctionnalités suivantes.

Packages modularisés

Les utilisateurs peuvent désormais utiliser un package distinct pour chaque service.

Nouvelle pile d'intergiciels

Les utilisateurs peuvent désormais utiliser une pile intergicielle pour contrôler le cycle de vie d'un appel d'opération.

De plus, le SDK est écrit TypeScript, ce qui présente de nombreux avantages, tels que le typage statique.

> **Important**
>
> Les exemples de code pour la v3 présentés dans ce guide sont écrits dans ECMAScript 6 (ES6). ES6apporte une nouvelle syntaxe et de nouvelles fonctionnalités pour rendre votre code plus moderne et lisible, et faire plus encore. ES6 nécessite que vous utilisiez Node.js version 13.x ou supérieure. Pour télécharger et installer la dernière version de Node.js, consultez la section Téléchargements de Node.js. Pour de plus amples informations, veuillez consulter JavaScript ES6Syntaxe /CommonJS.

Packages modularisés

La version 2 du SDK pour JavaScript (v2) exigeait que vous utilisiez l'intégralité du AWS SDK, comme suit.

```
var AWS = require("aws-sdk");
```

Le chargement de l'intégralité du SDK n'est pas un problème si votre application utilise de nombreux AWS services. Toutefois, si vous n'avez besoin que de quelques AWS services, cela signifie augmenter la taille de votre application avec du code dont vous n'avez pas besoin ou que vous n'utilisez pas.

Dans la version 3, vous pouvez charger et utiliser uniquement les AWS services individuels dont vous avez besoin. Cela est illustré dans l'exemple suivant, qui vous donne accès à Amazon DynamoDB (DynamoDB).

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";
```

Vous pouvez non seulement charger et utiliser AWS des services individuels, mais vous pouvez également charger et utiliser uniquement les commandes de service dont vous avez besoin. Cela est illustré dans les exemples suivants, qui vous permettent d'accéder au client DynamoDB et à la commande. `ListTablesCommand`

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
```

Important

Vous ne devez pas importer de sous-modules dans des modules. Par exemple, le code suivant peut générer des erreurs.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/
CognitoIdentity";
```

Le code suivant est correct.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
```

Comparaison de la taille du code

Dans la version 2 (v2), un exemple de code simple répertoriant toutes vos tables Amazon DynamoDB de `us-west-2` la région peut ressembler à ce qui suit.

```
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Tables names are ", data.TableNames);
  }
});
```

La v3 ressemble à ce qui suit.

```
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)
}
```

Le `aws-sdk` package ajoute environ 40 Mo à votre application. Le remplacement `var AWS = require("aws-sdk")` par `import {DynamoDB} from "@aws-sdk/client-dynamodb"` réduit cette surcharge à environ 3 Mo. Le fait de limiter l'importation au client `ListTablesCommand` et à la commande DynamoDB permet de réduire la surcharge à moins de 100 Ko.

```
// Load the DynamoDB client and ListTablesCommand command for Node.js
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbclient = new DynamoDBClient({});
```

Appeler des commandes dans la version 3

Vous pouvez effectuer des opérations dans la version 3 à l'aide des commandes v2 ou v3. Pour utiliser les commandes v3, vous importez les commandes et les clients du package AWS Services requis, puis exécutez la commande en utilisant la `.send` méthode utilisant le modèle async/await.

Pour utiliser les commandes v2, vous importez les packages de AWS services requis et exécutez la commande v2 directement dans le package à l'aide d'un modèle de rappel ou d'un modèle async/await.

Utilisation des commandes v3

La version 3 fournit un ensemble de commandes pour chaque package de AWS service afin de vous permettre d'effectuer des opérations pour ce AWS service. Après avoir installé un AWS service, vous pouvez parcourir les commandes disponibles dans le `node-modules/@aws-sdk/client-`*`PACKAGE_NAME`*`/commands folder.`

Vous devez importer les commandes que vous souhaitez utiliser. Par exemple, le code suivant charge le service DynamoDB et la commande. `CreateTableCommand`

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

Pour appeler ces commandes selon le modèle async/await recommandé, utilisez la syntaxe suivante.

```
CLIENT.send(new XXXCommand);
```

Par exemple, l'exemple suivant crée une table DynamoDB en utilisant le modèle async/await recommandé.

```
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);
};
```

Utilisation des commandes v2

Pour utiliser les commandes v2 dans le SDK pour JavaScript, vous devez importer les packages de AWS service complets, comme illustré dans le code suivant.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

Pour appeler des commandes v2 selon le modèle async/await recommandé, utilisez la syntaxe suivante.

```
client.command(parameters);
```

L'exemple suivant utilise la `createTable` commande v2 pour créer une table DynamoDB en utilisant le modèle async/await recommandé.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
const dynamoDB = new DynamoDB({ region: 'us-west-2' });
var tableParams = {
  TableName: TABLE_NAME
};
async function run() => {
  try {
    const data = await dynamoDB.createTable(tableParams);
    console.log("Success", data);
  }
  catch (err) {
    console.log("Error", err);
  }
};
run();
```

L'exemple suivant utilise la `createBucket` commande v2 pour créer un compartiment Amazon S3 à l'aide du modèle de rappel.

```
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();
```

Nouvelle pile d'intergiciels

La version v2 du SDK vous a permis de modifier une demande au cours des différentes étapes de son cycle de vie en y attachant des écouteurs d'événements. Cette approche peut rendre difficile le débogage de ce qui s'est mal passé au cours du cycle de vie d'une demande.

Dans la version 3, vous pouvez utiliser une nouvelle pile intergicielle pour contrôler le cycle de vie d'un appel d'opération. Cette approche présente quelques avantages. Chaque étape de middleware de la pile appelle l'étape de middleware suivante après avoir apporté des modifications à l'objet de la requête. Cela facilite également le débogage des problèmes dans la pile, car vous pouvez voir exactement quelles étapes du middleware ont été appelées à l'origine de l'erreur.

L'exemple suivant ajoute un en-tête personnalisé à un client Amazon DynamoDB (que nous avons créé et présenté précédemment) à l'aide d'un intergiciel. Le premier argument est une fonction qui accepte`next`, qui est la prochaine étape du middleware à appeler dans la pile`context`, et qui est un objet contenant des informations sur l'opération appelée. La fonction renvoie une fonction qui accepte`args`, c'est-à-dire un objet contenant les paramètres transmis à l'opération et à la demande. Il renvoie le résultat de l'appel du prochain intergiciel avec. `args`

```
dbclient.middlewareStack.add(
  (next, context) => args => {
    args.request.headers["Custom-Header"] = "value";
    return next(args);
  },
  {
    name: "my-middleware",
    override: true,
    step: "build"
  }
);

dbclient.send(new PutObjectCommand(params));
```

Quelle est la différence entre la AWS SDK pour JavaScript v2 et la v3

Cette section décrit les changements notables entre la AWS SDK pour JavaScript v2 et la v3. La v3 étant une réécriture modulaire de la v2, certains concepts de base sont différents entre la v2 et la v3. Vous pouvez prendre connaissance de ces changements dans nos articles de blog. Les articles de blog suivants vous permettront de vous tenir au courant :

- Packages modulaires en AWS SDK pour JavaScript
- Présentation de Middleware Stack en mode modulaire AWS SDK pour JavaScript

Le résumé des modifications d'interface de la AWS SDK pour JavaScript v2 à la v3 est donné ci-dessous. L'objectif est de vous aider à trouver facilement les équivalents v3 de la v2 APIs que vous connaissez déjà.

Rubriques

Constructeurs clients

Cette liste est indexée par les paramètres de configuration de la v2.

- `computeChecksums`
  - v2 : s'il faut calculer les MD5 sommes de contrôle pour les corps de charge utile lorsque le service les accepte (actuellement pris en charge dans S3 uniquement).
  - v3 : les commandes applicables de S3 (PutObject, PutBucketCors, etc.) calculeront automatiquement les MD5 sommes de contrôle de la charge utile de la demande. Vous pouvez également spécifier un algorithme de somme de contrôle différent dans le `ChecksumAlgorithm` paramètre des commandes pour utiliser un autre algorithme de somme de contrôle. Vous trouverez de plus amples informations dans l'annonce de la fonctionnalité S3.
- `convertResponseTypes`
  - v2 : Si les types sont convertis lors de l'analyse des données de réponse.
  - v3 : Obsolète. Cette option n'est pas considérée comme sûre car elle ne convertit pas les types tels que l'horodatage ou les binaires base64 à partir de la réponse JSON.
- `correctClockSkew`
  - v2 : s'il faut appliquer une correction d'inclinaison de l'horloge et réessayer les demandes qui échouent en raison d'une horloge client asymétrique.
  - v3 : Obsolète. Le SDK applique toujours une correction de l'inclinaison de l'horloge.
- `systemClockOffset`
  - v2 : valeur de décalage en millisecondes à appliquer à toutes les heures de signature.
  - v3 : Pas de changement.

- `credentials`
  - v2 : Les AWS informations d'identification avec lesquelles signer les demandes.
  - v3 : Pas de changement. Il peut également s'agir d'une fonction asynchrone qui renvoie des informations d'identification. Si la fonction renvoie un`expiration (Date)`, elle sera appelée à nouveau à l'approche de la date d'expiration. Consultez la référence de l'API v3 pour les `AwsAuthInputConfig` informations d'identification.
- `endpointCacheSize`
  - v2 : taille du cache global stockant les points de terminaison issus des opérations de découverte des points de terminaison.
  - v3 : Pas de changement.
- `endpointDiscoveryEnabled`
  - v2 : S'il faut appeler des opérations avec des points de terminaison fournis par le service de manière dynamique.
  - v3 : Pas de changement.
- `hostPrefixEnabled`
  - v2 : s'il faut associer les paramètres de demande au préfixe du nom d'hôte.
  - v3 : Obsolète. Le SDK injecte toujours le préfixe du nom d'hôte lorsque cela est nécessaire.
- `httpOptions`

  Ensemble d'options à transmettre à la requête HTTP de bas niveau. Ces options sont agrégées différemment dans la version 3. Vous pouvez les configurer en fournissant un nouveau`requestHandler`. Voici un exemple de définition des options http dans l'environnement d'exécution de Node.js. Vous trouverez plus d'informations dans la référence de l'API v3 pour NodeHttpHandler.

  Toutes les requêtes v3 utilisent le protocole HTTPS par défaut. Il vous suffit de fournir un HttpAgent personnalisé.

```
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*/
```

```
    }),
});
```

Si vous transmettez un point de terminaison personnalisé qui utilise http, vous devez fournir 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",
});
```

Si le client s'exécute dans des navigateurs, un ensemble d'options différent est disponible. Vous trouverez plus d'informations dans la référence de l'API v3 pour FetchHttpHandler.

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
const dynamodbClient = new DynamoDBClient({
    requestHandler: new FetchHttpHandler({
        requestTimeout: /* number in milliseconds */
    }),
});
```

Chaque option `httpOptions` est spécifiée ci-dessous :

- `proxy`
  - v2 : URL par laquelle les requêtes proxy doivent être transmises.
  - v3 : Vous pouvez configurer un proxy avec un agent en suivant la procédure Configuration des proxys pour Node.js.
- `agent`
  - v2 : L'objet Agent avec lequel effectuer des requêtes HTTP. Utilisé pour le regroupement de connexions.
  - v3 : Vous pouvez configurer `httpAgent` ou `httpsAgent` comme indiqué dans les exemples ci-dessus.
- `connectTimeout`

    - v2 : définit le socket pour qu'il expire après avoir échoué à établir une connexion avec le serveur au bout de quelques `connectTimeout` millisecondes.
    - v3 : `connectionTimeout` est disponible en `NodeHttpHandler` options.
  - `timeout`
    - v2 : Le nombre de millisecondes qu'une demande peut prendre avant d'être automatiquement résiliée.
    - v3 : `socketTimeout` est disponible en `NodeHttpHandler` options.
  - `xhrAsync`
    - v2 : Si le SDK enverra des requêtes HTTP asynchrones.
    - v3 : Obsolète. Les demandes sont toujours asynchrones.
  - `xhrWithCredentials`
    - v2 : Définit la propriété « WithCredentials » d'un objet XMLHttp Request.
    - v3 : Non disponible. Le SDK hérite des configurations de récupération par défaut.
- `logger`
  - v2 : un objet qui répond `.write()` (comme un flux) ou `.log()` (comme l'objet de console) afin de consigner des informations sur les demandes.
  - v3 : Pas de changement. Des journaux plus détaillés sont disponibles dans la version 3.
- `maxRedirects`
  - v2 : Le nombre maximum de redirections à suivre pour une demande de service.
  - v3 : Obsolète. Le SDK ne suit pas les redirections pour éviter les demandes interrégionales involontaires.
- `maxRetries`
  - v2 : nombre maximal de tentatives à effectuer pour une demande de service.
  - v3 : Remplacé en. `maxAttempts` Pour en savoir plus, consultez la référence de l'API v3 pour RetryInputConfig. Notez que cela `maxAttempts` devrait être le cas`maxRetries + 1`.
- `paramValidation`
  - v2 : Si les paramètres d'entrée doivent être validés par rapport à la description de l'opération avant d'envoyer la demande.
  - v3 : Obsolète. Le SDK n'effectue pas de validation côté client lors de l'exécution.
- `region`
  - v2 : région à laquelle envoyer les demandes de service.
  - v3 : Pas de changement. Il peut également s'agir d'une fonction asynchrone qui renvoie une chaîne de région.

- `retryDelayOptions`

  - v2 : ensemble d'options permettant de configurer le délai de nouvelle tentative en cas d'erreur réessayable.
  - v3 : Obsolète. Le SDK prend en charge une stratégie de nouvelle tentative plus flexible avec l'option `retryStrategy` client constructeur. Pour en savoir plus, consultez la référence de l'API v3.
- `s3BucketEndpoint`
  - v2 : si le point de terminaison fourni adresse un compartiment individuel (faux s'il s'adresse au point de terminaison de l'API racine).
  - v3 : Remplacé en. `bucketEndpoint` Pour en savoir plus, consultez la référence de l'API v3 pour BucketEndpoint. Notez que lorsque ce paramètre est défini sur`true`, vous spécifiez le point de terminaison de la `Bucket` demande dans le paramètre de demande, le point de terminaison d'origine sera remplacé. Alors que dans la version v2, le point de terminaison de la demande dans le constructeur du client remplace le paramètre de `Bucket` demande.
- `s3DisableBodySigning`
  - v2 : S'il faut désactiver la signature du corps S3 lors de l'utilisation de la version de signature v4.
  - v3 : renommé en. `applyChecksum`
- `s3ForcePathStyle`
  - v2 : s'il faut forcer le style de chemin URLs pour les objets S3.
  - v3 : renommé en. `forcePathStyle`
- `s3UseArnRegion`
  - v2 : S'il faut remplacer la région de demande par la région déduite de l'ARN de la ressource demandée.
  - v3 : renommé en. `useArnRegion`
- `s3UsEast1RegionalEndpoint`
  - v2 : Lorsque la région est définie sur « us-east-1 », s'il faut envoyer une requête s3 aux points de terminaison globaux ou aux points de terminaison régionaux « us-east-1 ».
  - v3 : Obsolète. Le client S3 utilisera toujours le point de terminaison régional si la région est définie sur`us-east-1`. Vous pouvez définir la région pour envoyer `aws-global` des demandes au point de terminaison global S3.
- `signatureCache`
  - v2 : si la signature avec laquelle signer les demandes (en remplacement de la configuration de l'API) est mise en cache.
  - v3 : Obsolète. Le SDK met toujours en cache les clés de signature hachées.
- `signatureVersion`

  - v2 : version de signature avec laquelle signer les demandes (en remplacement de la configuration de l'API).
  - v3 : Obsolète. La signature V2 prise en charge dans le SDK v2 a été déconseillée par AWS. La v3 ne prend en charge que la signature v4.
- `sslEnabled`
  - v2 : si le protocole SSL est activé pour les demandes.
  - v3 : renommé en. `tls`
- `stsRegionalEndpoints`
  - v2 : S'il faut envoyer la demande sts aux points de terminaison mondiaux ou aux points de terminaison régionaux.
  - v3 : Obsolète. Le client STS utilisera toujours des points de terminaison régionaux s'il est défini sur une région spécifique. Vous pouvez définir la région pour envoyer une demande `aws-global` au point de terminaison global STS.
- `useAccelerateEndpoint`
  - v2 : s'il faut utiliser le point de terminaison Accelerate avec le service S3.
  - v3 : Pas de changement.

Fournisseurs d'informations d'identification

Dans la version 2, le SDK pour JavaScript fournit une liste de fournisseurs d'informations d'identification parmi lesquels choisir, ainsi qu'une chaîne de fournisseurs d'informations d'identification, disponible par défaut sur Node.js, qui essaie de charger les AWS informations d'identification de tous les fournisseurs les plus courants. Le SDK pour JavaScript v3 simplifie l'interface du fournisseur d'informations d'identification, ce qui facilite l'utilisation et la rédaction de fournisseurs d'informations d'identification personnalisés. En plus d'une nouvelle chaîne de fournisseurs d'informations d'identification, le SDK pour la JavaScript version 3 fournit tous une liste de fournisseurs d'informations d'identification visant à fournir un équivalent à la version 2.

Voici tous les fournisseurs d'identifiants de la version 2 et leurs équivalents de la version 3.

Fournisseur d'informations d'identification par défaut

Le fournisseur d'informations d'identification par défaut est la façon dont le SDK permet de JavaScript résoudre les AWS informations d'identification si vous ne les fournissez pas explicitement.

- v2 : CredentialProviderChaindans Node.js, résout les informations d'identification provenant des sources dans l'ordre suivant :

- Variable environnementale
- Fichier d'informations d'identification partagé
- Informations d'identification du conteneur ECS
- Processus externe de génération
- Jeton OIDC provenant du fichier spécifié
- Métadonnées de EC2 l'instance Amazon

Si l'un des fournisseurs d'identifiants ci-dessus ne parvient pas à résoudre l' AWS identifiant, la chaîne revient au fournisseur suivant jusqu'à ce qu'un identifiant valide soit résolu, et la chaîne génère une erreur lorsque tous les fournisseurs échouent.

Dans les environnements d'exécution Browser et React Native, la chaîne d'informations d'identification est vide et les informations d'identification doivent être définies explicitement.

- v3 : DefaultProvider. Les sources d'informations d'identification et l'ordre de repli ne changent pas dans la version 3. Il prend également en charge les AWS IAM Identity Center informations d'identification.

Informations d'identification temporaires

- v2 : `ChainableTemporaryCredentials`représente les informations d'identification temporaires extraites de`AWS.STS`. Sans aucun paramètre supplémentaire, les informations d'identification seront extraites de l'`AWS.STS.getSessionToken()`opération. Si un rôle IAM est fourni, l'`AWS.STS.assumeRole()`opération sera plutôt utilisée pour récupérer les informations d'identification du rôle. `AWS.ChainableTemporaryCredentials`diffère de la `AWS.TemporaryCredentials` façon dont les MasterCredentials et les actualisations sont gérés. `AWS.ChainableTemporaryCredentials`actualise les informations d'identification expirées à l'aide des MasterCredentials transmises par l'utilisateur pour permettre le chaînage des informations d'identification STS. Cependant, les informations d'identification principales sont réduites de `AWS.TemporaryCredentials` manière récursive lors de l'instanciation, ce qui empêche d'actualiser les informations d'identification qui nécessitent des informations d'identification temporaires intermédiaires.

  L'original `TemporaryCredentials`a été déprécié au profit de la `ChainableTemporaryCredentials` version v2.

- version 3 : `fromTemporaryCredentials`. Vous pouvez appeler `fromTemporaryCredentials()` depuis le `@aws-sdk/credential-providers` forfait. Voici un exemple :

```
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 },
  }),
});
```

Informations d'identification Amazon Cognito

Chargez les informations d'identification depuis le service Amazon Cognito Identity, normalement utilisé dans les navigateurs.

- v2 : `CognitoIdentityCredentials`représente les informations d'identification récupérées auprès de STS Web Identity Federation à l'aide du service Amazon Cognito Identity.
- v3 : `Cognito Identity Credential Provider`Le `@aws/credential-providers`package fournit deux fonctions de fournisseur d'informations d'identification, dont l'une `fromCognitoIdentity`prend un identifiant d'identité et des appels`cognitoIdentity:GetCredentialsForIdentity`, tandis que l'autre `fromCognitoIdentityPool`prend un identifiant de pool d'identités, appelle `cognitoIdentity:GetId` lors du premier appel, puis appelle. `fromCognitoIdentity` Les invocations ultérieures de ce dernier ne sont pas réinvoquées. GetId

  Le fournisseur met en œuvre le « flux simplifié » décrit dans le guide du développeur Amazon Cognito. Le « flux classique » qui implique d'appeler `cognito:GetOpenIdToken` puis d'appeler n'*`sts:AssumeRoleWithWebIdentity`*est pas pris en charge. Veuillez nous envoyer une demande de fonctionnalité si vous en avez besoin.

```
// 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",
    },
  }),
});
```

Identifiant Amazon EC2 Metadata (IMDS)

Représente les informations d'identification reçues du service de métadonnées sur une EC2 instance Amazon.

- version 2 : `EC2MetadataCredentials`
- version 3 : `fromInstanceMetadata`. Crée un fournisseur d'informations d'identification qui fournira les informations d'identification auprès du service de métadonnées d' EC2 instance Amazon.

```
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
  }),
});
```

Informations d'identification Amazon ECS

Représente les informations d'identification reçues depuis l'URL spécifiée. Ce fournisseur demandera des informations d'identification temporaires à partir de l'URI spécifiée par la variable d'environnement `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` ou par la variable d'`AWS_CONTAINER_CREDENTIALS_FULL_URI`environnement.

- v2 : `ECSCredentials` ou `RemoteCredentials`
- version 3 : `fromContainerMetadata`. Crée un fournisseur d'informations d'identification qui fournira les informations d'identification auprès du service de métadonnées de conteneur Amazon ECS.

```
import { fromContainerMetadata } from "@aws-sdk/credential-providers"; // ES6 import

const client = new FooClient({
  credentials: fromContainerMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
```

```
  }),
});
```

Informations d'identification du système de fichiers

- v2 : `FileSystemCredentials`. Représente les informations d'identification d'un fichier JSON sur le disque.
- v3 : Obsolète. Vous pouvez lire explicitement le fichier JSON et le fournir au client. Veuillez nous envoyer une demande de fonctionnalité si vous en avez besoin.

Fournisseur d'informations d'identification SAML

- v2 : `SAMLCredentials`représente les informations d'identification extraites du support STS SAML.
- v3 : Non disponible. Veuillez nous envoyer une demande de fonctionnalité si vous en avez besoin.

Informations d'identification partagées Informations d'identification

Charge les informations d'identification à partir du fichier d'informations d'identification partagé (défini par défaut `~/.aws/credentials` ou défini par la variable d'`AWS_SHARED_CREDENTIALS_FILE`environnement). Ce fichier est pris en charge par différents AWS SDKs outils. Vous pouvez consulter le document sur les fichiers de configuration et d'identification partagés pour plus d'informations.

- version 2 : `SharedIniFileCredentials`
- version 3 : `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
  }),
});
```

Identifiants d'identité Web

Récupère les informations d'identification à l'aide d'un jeton OIDC à partir d'un fichier sur le disque. Couramment utilisé dans Amazon EKS.

- version 2 : TokenFileWebIdentityCredentials
- version 3 : 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 },
  }),
});
```

Informations d'identification de la Fédération des identités Web

Récupère les informations d'identification auprès du support de fédération d'identité Web STS.

- version 2 : WebIdentityCredentials
- version 3 : 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 },
  }),
});
```

Considérations relatives à Amazon S3

Chargement en plusieurs parties sur Amazon S3

Dans la version 2, le client Amazon S3 contient une `upload()`opération qui prend en charge le téléchargement d'objets volumineux grâce à la fonctionnalité de téléchargement partitionné proposée par Amazon S3.

Dans la version 3, le `@aws-sdk/lib-storage`package est disponible. Il prend en charge toutes les fonctionnalités proposées dans l'`upload()`opération v2 et supporte à la fois Node.js et le runtime des navigateurs.

URL présignée Amazon S3

Dans la version 2, le client Amazon S3 contient les `getSignedUrlPromise()`opérations `getSignedUrl()`et pour générer une URL que les utilisateurs peuvent utiliser pour charger ou télécharger des objets depuis Amazon S3.

Dans la version 3, le `@aws-sdk/s3-request-presigner`package est disponible. Ce paquet contient les fonctions à la fois pour les `getSignedUrlPromise()` opérations `getSignedUrl()` et pour les opérations. Ce billet de blog décrit les détails de ce package.

Redirections régionales Amazon S3

Si une région incorrecte est transmise au client Amazon S3 et qu'une erreur ultérieure `PermanentRedirect` (statut 301) est renvoyée, le client Amazon S3 de la version 3 prend en charge les redirections régionales (anciennement connu sous le nom de client global Amazon S3 dans la version 2). Vous pouvez utiliser l'`followRegionRedirects`indicateur dans la configuration

du client pour que le client Amazon S3 suive les redirections régionales et prenne en charge sa fonction de client global.

> **Note**
>
> Notez que cette fonctionnalité peut entraîner une latence supplémentaire car les demandes ayant échoué sont réessayées avec une région corrigée lors de la réception `PermanentRedirect` d'une erreur avec le statut 301. Cette fonctionnalité ne doit être utilisée que si vous ne connaissez pas à l'avance la région de vos compartiments.

Streaming Amazon S3 et réponses mises en mémoire tampon

Le SDK v3 préfère ne pas mettre en mémoire tampon les réponses potentiellement volumineuses. Cela se produit fréquemment lors de l'`GetObject`opération Amazon S3, qui a renvoyé un `Buffer` dans la version 2, mais renvoie un `Stream` dans la version 3.

Pour Node.js, vous devez utiliser le flux ou la collecte des déchets du client ou de son gestionnaire de requêtes pour maintenir les connexions ouvertes au nouveau trafic en libérant des sockets.

```
// 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.
```

Pour plus d'informations, consultez la section sur l'épuisement des sockets.

Client de documents DynamoDB

Utilisation de base du client de documents DynamoDB dans la version 3

- Dans la version 2, vous pouvez utiliser la `AWS.DynamoDB.DocumentClient`classe pour appeler APIs DynamoDB avec des types JavaScript natifs tels que Array, Number et Object. Il simplifie ainsi l'utilisation des éléments dans Amazon DynamoDB en éliminant la notion de valeurs d'attribut.
- Dans la version 3, le `@aws-sdk/lib-dynamodb`client équivalent est disponible. Il est similaire aux clients de service normaux du SDK v3, à la différence près qu'il prend un client DynamoDB de base dans son constructeur.

Exemple :

```
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`valeurs saisies lors du triage

- Dans la version 2, `undefined` les valeurs des objets étaient automatiquement omises lors du processus de compilation vers DynamoDB.

- Dans la version 3, le comportement de triage par défaut `@aws-sdk/lib-dynamodb` a changé : les objets contenant des `undefined` valeurs ne sont plus omis. Pour s'aligner sur les fonctionnalités de la version 2, les développeurs doivent définir explicitement le `removeUndefinedValues` to `true` dans le `marshallOptions` client de documents DynamoDB.

Exemple :

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

// The DynamoDBDocumentClient is configured to handle undefined values properly
const ddbDocClient = DynamoDBDocumentClient.from(client, {
  marshallOptions: {
    removeUndefinedValues: true
  }
});

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "123",
      content: undefined // This value will be automatically omitted.
      array: [1, undefined], // The undefined value will be automatically omitted.
      map: { key: undefined }, // The "key" will be automatically omitted.
      set: new Set([1, undefined]), // The undefined value will be automatically
 omitted.
    };
  })
);
```

D'autres exemples et configurations sont disponibles dans le package README.

Serveurs et signataires

Cette page décrit l'utilisation des serveurs et des signataires dans la AWS SDK pour JavaScript version 3.

Programmes d'attente

Dans la version 2, tous les serveurs sont liés à la classe du client de service, et vous devez spécifier dans la saisie du serveur l'état conçu que le client attendra. Par exemple, vous devez appeler `waitFor("bucketExists")`pour attendre qu'un compartiment nouvellement créé soit prêt.

Dans la version 3, il n'est pas nécessaire d'importer des serveurs si votre application n'en a pas besoin. De plus, vous ne pouvez importer que le serveur dont vous avez besoin pour attendre l'état souhaité. Ainsi, vous pouvez réduire la taille de votre offre groupée et améliorer les performances. Voici un exemple d'attente pour que le bucket soit prêt après sa création :

```
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 });
```

Vous trouverez tout sur la façon de configurer les serveurs dans le billet de blog sur les serveurs dans la AWS SDK pour JavaScript v3.

Amazon CloudFront Signer

Dans la version 2, vous pouvez signer la demande d'accès aux CloudFront distributions Amazon restreintes avec `AWS.CloudFront.Signer`.

Dans la version 3, vous disposez des mêmes utilitaires fournis dans le `@aws-sdk/cloudfront-signer`package.

Signataire Amazon RDS

Dans la version 2, vous pouvez générer le jeton d'authentification vers une base de données Amazon RDS à l'aide de. `AWS.RDS.Signer`

Dans la version 3, la classe utilitaire similaire est disponible dans le `@aws-sdk/rds-signer`package.

Signataire d'Amazon Polly

Dans la version 2, vous pouvez générer une URL signée vers le discours synthétisé par le service Amazon Polly avec. `AWS.Polly.Presigner`

Dans la version 3, la fonction utilitaire similaire est disponible dans le `@aws-sdk/polly-request-presigner`package.

Remarques sur des clients de services spécifiques

AWS Lambda

Le type de réponse aux appels Lambda diffère en v2 et en v3.

```
// v2
import { Lambda } from "@aws-sdk/client-lambda";
import AWS from "aws-sdk";

const lambda = new AWS.Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
}).promise();

// in v2, Lambda::invoke::Payload is automatically converted to string via a
// specific code customization.
const payloadIsString = typeof invoke.Payload === "string";
console.log("Invoke response payload type is string:", payloadIsString);

const payloadObject = JSON.parse(invoke.Payload);
console.log("Invoke response object", payloadObject);
```

```
// v3
const lambda = new Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
});

// in v3, Lambda::invoke::Payload is not automatically converted to a string.
// This is to reduce the number of customizations that create inconsistent behaviors.
const payloadIsByteArray = invoke.Payload instanceof Uint8Array;
```

```
console.log("Invoke response payload type is Uint8Array:", payloadIsByteArray);

// To maintain the old functionality, only one additional method call is needed:
// v3 adds a method to the Uint8Array called transformToString.
const payloadObject = JSON.parse(invoke.Payload.transformToString());
console.log("Invoke response object", payloadObject);
```

Amazon SQS

MD5 Somme de contrôle

Pour ignorer le calcul des MD5 checksums des corps des messages, définissez `md5` la valeur false sur l'objet de configuration. Sinon, le SDK calculera par défaut la somme de contrôle pour l'envoi de messages et validera la somme de contrôle pour les messages récupérés.

```
// 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
});
```

Lors de l'utilisation d'une personnalisation `QueueUrl` dans les opérations Amazon SQS qui l'ont comme paramètre d'entrée, dans la version v2, il était possible de fournir une personnalisation `QueueUrl` qui remplacerait le point de terminaison par défaut du client Amazon SQS.

Messages multirégionaux

Vous devez utiliser un client par région dans la version 3. La AWS région est destinée à être initialisée au niveau du client et à ne pas être modifiée entre les demandes.

```
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);
}
```

Point de terminaison personnalisé

Dans la version 3, lorsque vous utilisez un point de terminaison personnalisé, c'est-à-dire un point de terminaison différent des points de terminaison publics Amazon SQS par défaut, vous devez toujours définir le point de terminaison sur le client Amazon SQS ainsi que sur le terrain. `QueueUrl`

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  // client endpoint should be specified in v3 when not the default public SQS endpoint
 for your region.
  // This is required for versions <= v3.506.0
  // This is optional but recommended for versions >= v3.507.0 (a warning will be
 emitted)
  endpoint: "https://my-custom-endpoint:8000/",
});

await sqs.sendMessage({
  QueueUrl: "https://my-custom-endpoint:8000/1234567/MyQueue",
  Message: "hello",
});
```

Si vous n'utilisez pas de point de terminaison personnalisé, vous n'avez pas besoin de configurer `endpoint` le client.

```
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",
});
```

Documentation supplémentaire

Le tableau suivant contient des liens vers de la documentation supplémentaire qui vous aidera à utiliser et à comprendre le AWS SDK pour JavaScript (v3).

Nom	Remarques
Clients du SDK	Informations sur l'initialisation d'un client SDK et sur les paramètres de constructeur configura bles courants.
Notes de mise à niveau (2.x vers 3.x)	Informations concernant la mise à niveau depuis AWS SDK pour JavaScript (v2).
Utilisation du AWS SDK pour JavaScript (v3) sur les environnements d'exécution de AWS Lambda Node.js	Meilleures pratiques pour travailler dans le cadre de AWS Lambda l'utilisation du AWS SDK pour JavaScript (v3).
Performances	Des informations sur la manière dont l'équipe du AWS SDK a optimisé les performances du SDK et des conseils pour configurer le SDK afin qu'il fonctionne efficacement.
TypeScript	TypeScript conseils et astuces FAQs liés à la AWS SDK pour JavaScript (v3).
Gestion des erreurs	Conseils pour gérer les erreurs liées au AWS SDK pour JavaScript (v3).
Pratiques efficaces	Recommandations générales pour l'utilisation du AWS SDK pour JavaScript (v3).

Historique du document pour AWS SDK pour JavaScript la version 3

Historique du document

Le tableau suivant décrit les modifications importantes apportées à la version V3 AWS SDK pour JavaScriptà compter du 20 octobre 2020. Pour recevoir les notifications concernant les mises à jour de cette documentation, abonnez-vous à un flux RSS.

Modification	Description	Date
Le protocole TLS 1.3 est désormais pris en charge sur tous les points de terminaison des API de AWS service dans toutes les régions	Version TLS prise en charge et méthode de journalisation de la version TLS mises à jour.	10 avril 2025
Génération de clients avec les @smithy /types	Contenu mis à jour avec la génération de clients à l'aide du package @smithy /types.	15 février 2025
Protection de l'intégrité des données avec des checksums	Contenu mis à jour avec des détails sur le calcul automatiq ue de la somme de contrôle.	15 janvier 2025
Sommes de contrôle Amazon S3	Ajout d'une section expliquan t comment utiliser des checksums flexibles avec Amazon S3.	1er janvier 2025
Support pour les points de terminaison basés sur des comptes dans DynamoDB	La prise en charge AWS SDK pour JavaScript supplémen taire des points de terminaison basés sur des comptes dans DynamoDB.	26 septembre 2024

Nouveau sujet sur la journalisation du SDK	Une rubrique décrivant comment enregistrer les appels d'API effectués avec le SDK pour JavaScript a été ajoutée, y compris des informations sur l'utilisation d'un intergiciel pour enregistrer les demandes.	26 septembre 2024
Annonce	Bannière supérieure mise à jour avec un end-of-support rappel pour Internet Explorer 11.	23 septembre 2022
Mises à jour mineures	Mises à jour mineures visant à clarifier et à résoudre les liens rompus. Ajout de liens de sensibilisation AWS SDKs et d'un guide de référence sur les outils.	22 août 2022
Appliquer une version minimale de TLS	Ajout d'informations sur le protocole TLS 1.3.	31 mars 2022
Mise à jour de la rubrique Définir les informations d'identification dans Node.js	Mettre à jour la rubrique concernant la définition des informations d'identification dans Node.js pour AWS SDK pour JavaScript V3.	20 octobre 2020
Migrer vers la version 3	Ajout d'une rubrique pour décrire comment migrer vers la AWS SDK pour JavaScript version 3.	20 octobre 2020

Commencer	Rubriques mises à jour pour démarrer dans le navigateur et utiliser Node.js pour le AWS SDK pour JavaScript V3.	20 octobre 2020
Générateur de navigateur	Les informations relatives à AWS Browser Builder ont été supprimées car elles ne sont pas requises pour la AWS SDK pour JavaScript version 3.	20 octobre 2020
Exemples de services Amazon Transcribe mis à jour	Exemples de services Amazon Transcribe mis à jour pour AWS SDK pour JavaScript la version 3.	20 octobre 2020
Exemples de services Amazon Simple Notification Service mis à jour	Exemples de services Amazon Simple Notification Service mis à jour pour la AWS SDK pour JavaScript version 3.	20 octobre 2020
Exemples de services Amazon Simple Email Service mis à jour	Exemples de services Amazon Simple Email Service mis à jour pour la AWS SDK pour JavaScript version 3.	20 octobre 2020
Exemples de services Amazon Redshift mis à jour	Exemples de services Amazon Redshift mis à jour pour AWS SDK pour JavaScript la version 3.	20 octobre 2020
Exemples de services Amazon Lex mis à jour	Exemples de services Amazon Lex mis à jour pour la AWS SDK pour JavaScript version 3.	20 octobre 2020

AWS Elemental MediaConvert exemples de services mis à jour	Exemples AWS Elemental MediaConvert de services mis à jour pour la AWS SDK pour JavaScript V3.	20 octobre 2020
AWS Lambda exemples de services mis à jour	Exemples AWS Lambda de services mis à jour pour la AWS SDK pour JavaScript V3.	20 octobre 2020
AWS SDK pour JavaScript Aperçu du guide du développeur V3	Publication de la version préliminaire du guide du développeur AWS SDK pour JavaScript V3.	19 octobre 2020