Guide du développeur pour la version 3 du SDK

AWS SDK for JavaScript

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

Table of Contents

..... viii
Qu'est-ce que c'est AWS SDK for JavaScript ? 1
Maintenance et prise en charge des versions majeures du SDK 1
Nouveautés de la version 3 2
Packages modularisés 2
Nouvelle pile de middleware 7
Utilisation du kit SDK avec Node.js 8
Utilisation du SDK avec AWS Cloud9 8
Utilisation du SDK avec AWS Amplify 8
Utilisation du SDK avec des navigateurs Web 8
Utilisation des navigateurs dans la V3 9
Cas d'utilisation courants 9
À propos des exemples 10
Ressources 10
Mise en route 11
Authentification du SDK avec AWS 11
Démarrage d’une session de portail d'accès AWS 12
Plus d'informations d'authentification 13
Commencez avec Node.js 14
Le scénario 14
Prérequis 14
Étape 1 : Configuration de la structure des packages et installation des packages clients 15
Étape 2 : ajouter les importations et le code SDK nécessaires 16
Étape 3 : Exécutez l'exemple 18
Commencez dans le navigateur 18
Scénario 19
Étape 1 : créer un pool d'identités et un rôle IAM Amazon Cognito 19
Étape 2 : ajouter une politique au rôle IAM créé 20
Étape 3 : ajouter un compartiment et un objet Amazon S3 21
Étape 4 : configurer le code du navigateur 22
Étape 5 : Exécuter l'exemple 23
Nettoyage 23
Configurer le SDK pour JavaScript 25
Prérequis 25

Configuration d'un environnement AWS Node.js 25
Navigateurs Web pris en charge 26
Installer le SDK 27
Chargez le SDK 28
Configurer le SDK pour JavaScript 29
Configuration par service 29
Définir la configuration par service 30
Définissez la AWS région 30
Dans un constructeur de classe client 31
Utiliser une variable d'environnement 31
Utiliser un fichier de configuration partagé 31
Ordre de priorité pour définir la région 32
Définir les informations d'identification 32
Bonnes pratiques en matière d'accréditation 32
Définissez les informations d'identification dans Node.js 33
Définir les informations d'identification dans un navigateur Web 37
Considérations relatives à Node.js 41
Utiliser les modules Node.js intégrés 41
Utiliser les packages npm 41
Configurer MaxSockets dans Node.js 42
Réutiliser les connexions avec keep-alive dans Node.js 43
Configurer les proxys pour Node.js 43
Enregistrez les ensembles de certificats dans Node.js 44
Considérations à propos des scripts du navigateur 45
Créez le SDK pour les navigateurs 45
Partage des ressources cross-origin (CORS) 46
Bundle avec webpack 50
Travaillez avec les AWS services 55
Création et appel d'objets de service 55
Spécifier les paramètres de l'objet de service 56
Appeler les services de manière asynchrone 56
Gérer les appels asynchrones 57
Utiliser async/await 58
Promesses d'utilisation 59
Utiliser une fonction de rappel 60
Création de demandes de service pour les clients 61

Gérer les réponses des clients du service ... 63
Données d'accès renvoyées dans la réponse ... 63
Informations sur les erreurs d'accès ... 63
Travailler avec JSON ... 63
JSON en tant que paramètres d'objet de service ... 64
Sous-ensemble d'exemples de code avec conseils ... 65
JavaScript Syntaxe ES6/CommonJS ... 66
Exemples d'Amazon DynamoDB ... 69
Exemples AWS Elemental MediaConvert ... 95
Exemples AWS Lambda ... 117
Exemples d'Amazon Lex ... 117
Exemples d'Amazon Polly ... 118
Exemples d'Amazon Redshift ... 121
Exemples Amazon SES ... 129
Exemples Amazon SNS ... 158
Exemples d'Amazon Transcribe ... 193
Cross-service : Configuration de Node.js sur une instance Amazon EC2 ... 204
Cross-service : application pour envoyer des données ... 207
Multiservice : application de transcription ... 215
Multiservice : Amazon API Gateway et Lambda ... 227
Multiservice : flux de travail sans serveur avec Step Functions ... 242
Cross-service : événements Lambda planifiés ... 258
Cross-service : exemple d'Amazon Lex ... 270
Multiservice : application de messagerie ... 283
À utiliser AWS Cloud9 avec le SDK pour JavaScript ... 297
Étape 1 : Configurez votre AWS compte pour utiliser AWS Cloud9 ... 297
Étape 2 : Configuration de votre environnement AWS Cloud9 de développement ... 298
Étape 3 : configurer le SDK pour JavaScript ... 298
Pour configurer le SDK JavaScript pour Node.js ... 298
Pour configurer le SDK JavaScript dans le navigateur ... 299
Étape 4 : Téléchargez un exemple de code ... 299
Étape 5 : Exécuter et déboguer un exemple de code ... 300
Exemples de code ... 301
Actions et scénarios ... 301
Auto Scaling ... 303
Amazon Bedrock ... 346

Amazon Bedrock Runtime 351
Agents for Amazon Bedrock 366
Agents pour Amazon Bedrock Runtime 380
CloudWatch 383
CloudWatch Évènements 399
CloudWatch Journaux 406
CodeBuild 424
Fournisseur d'identité Amazon Cognito 428
DynamoDB 448
Amazon EC2 495
Elastic Load Balancing 578
EventBridge 628
AWS Glue 635
HealthImaging 660
IAM 697
Lambda 809
Amazon Personalize 819
Amazon Personalize Events 837
Amazon Personalize Runtime 841
Amazon Pinpoint 846
Amazon Redshift 856
Amazon S3 861
S3 Glacier 900
SageMaker 904
Secrets Manager 937
Amazon SES 939
Amazon SNS 963
Amazon SQS 1002
Step Functions 1039
AWS STS 1041
AWS Support 1044
Amazon Transcribe 1062
Exemples de services croisés 1071
Créer une application Amazon Transcribe 1072
Créer une application de streaming Amazon Transcribe 1073
Créer une application pour soumettre des données à une table DynamoDB 1073

Création d'un chatbot Amazon Lex 1074
Création d’une application sans serveur pour gérer des photos 1074
Créer une application web pour suivre les données DynamoDB 1075
Créer un outil de suivi des éléments de travail sans serveur Aurora 1075
Créer une application Amazon Textract Explorer 1076
Créez une application pour analyser les commentaires des clients 1076
Détecter l'EPI dans des images 1080
Détecter des objets dans des images 1081
Détecter des personnes et des objets dans une vidéo 1082
Appeler une fonction Lambda à partir d'un navigateur 1082
Utiliser API Gateway pour invoquer une fonction Lambda 1083
Utiliser les fonctions Step Functions pour appeler des fonctions Lambda 1084
Utilisent des événements planifiés pour invoquer une fonction Lambda 1084
Sécurité 1086
Protection des données 1086
Gestion des identités et des accès 1088
Public ciblé 1088
Authentification par des identités 1089
Gestion des accès à l’aide de politiques 1093
Comment Services AWS travailler avec IAM 1095
Résolution des problèmes AWS d'identité et d'accès 1096
Validation de la conformité 1098
Résilience 1099
Sécurité de l’infrastructure 1100
Appliquer une version TLS minimale 1100
Vérifier et appliquer TLS dans Node.js 1101
Vérifier et appliquer TLS dans un script de navigateur 1103
Migrer vers la version 3 1106
Migrer vers la V3 1106
Utiliser codemod pour migrer le code v2 existant 1106
Historique de la documentation 1108
Historique du document 1108

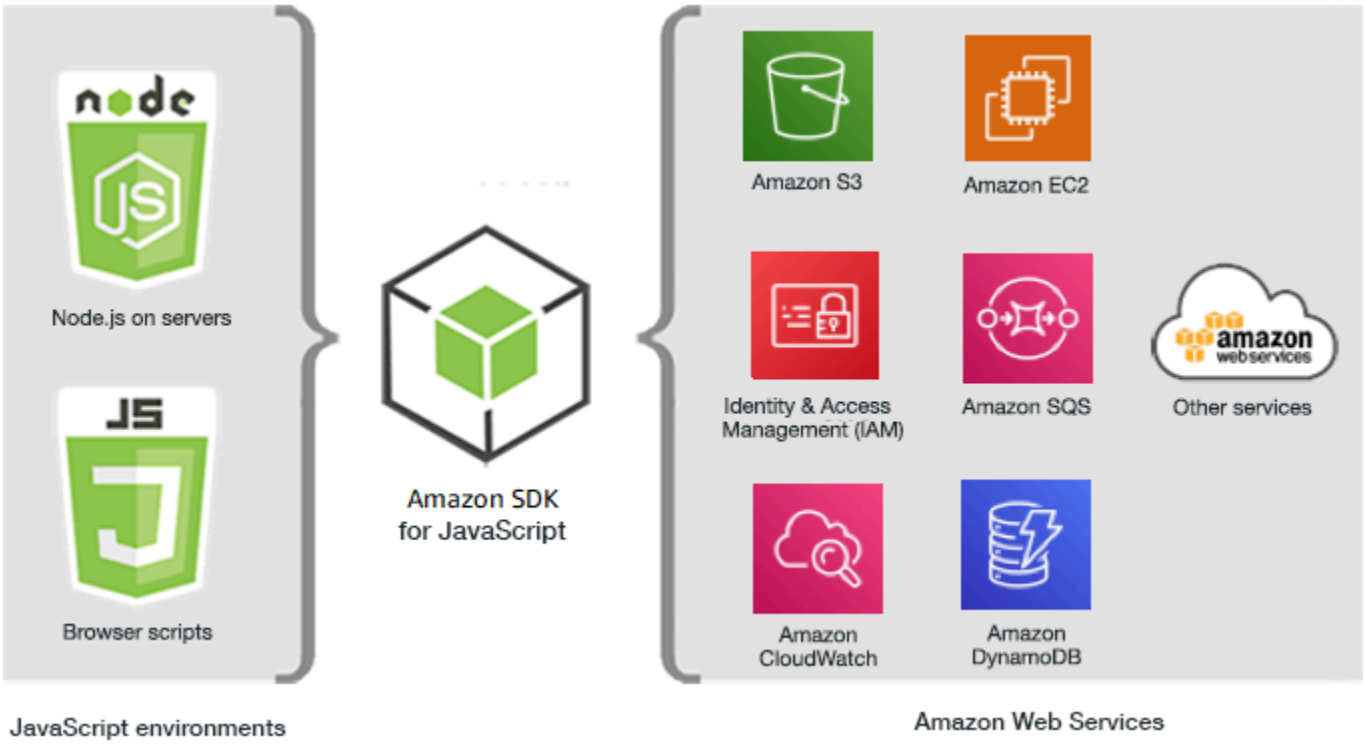
Le guide de référence de l'API AWS SDK for JavaScript V3 décrit en détail toutes les opérations de l'API pour la AWS SDK for JavaScript version 3 (V3).

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 for JavaScript ?

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

Le guide de référence de l'API AWS SDK for 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.



Maintenance et prise en charge des versions majeures du SDK

Pour en savoir plus sur la maintenance et la prise en charge des versions majeures du SDK et de leurs dépendances sous-jacentes, consultez la section suivante dans le AWSGuide de référence des kits SDK et des outils :

- Politique de maintenance des kits SDK et des outils AWS
- Matrice de prise en charge des versions des kits SDK et des outils AWS

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 de middleware

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 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. Pour plus d'informations, veuillez consulter JavaScript Syntaxe ES6/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 ne pouvez charger et utiliser que 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";
(async function () {
   const dbclient = new DynamoDBClient({ region: 'us-west-2'});

  try {
    const results = await dbclient.send(new ListTablesCommand);
    results.TableNames.forEach(function (item, index) {
      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 charge 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 V3

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'});
var tableParams = {
    Table : TABLE_NAME
};
(async function () => {
      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 les 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 dymamoDB = new DynamoDB({region: 'us-west-2'});
var tableParams = {
    TableName : TABLE_NAME
};
async function run() => {
      try {
            const data = await dymamoDB.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);
         }
    })
};
```

Nouvelle pile de middleware

La version 2 du SDK vous a permis de modifier une demande au cours des différentes étapes de son cycle de vie en y associant des écouteurs d'événements. Cette approche peut rendre difficile le débogage des erreurs survenues 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);
  },
  {
    step: "build"
  }
);
```

```
dbclient.send(new PutObjectCommand(params));
```

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 fonctions AWS Lambda 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'API spécifiques diffère entre Node.js et le navigateur, nous signalons ces différences.

Utilisation du SDK avec AWS Cloud9

Vous pouvez également développer des applications Node.js à l'aide du SDK pour JavaScript l'AWS Cloud9IDE. Pour plus d'informations sur AWS Cloud9 l'utilisation du SDK pour JavaScript, consultezÀ utiliser AWS Cloud9 avec AWS SDK for JavaScript.

Utilisation du SDK avec AWS Amplify

Pour les applications Web, mobiles et hybrides basées sur un navigateur, vous pouvez également utiliser la AWS Amplifybibliothèque 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 une liste des navigateurs pris en charge par le kit AWS SDK for JavaScript, consultez Navigateurs 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'API spécifiques 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 for 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.
- Utilisez AWS Lambda pour encapsuler la logique propriétaire que vous pouvez appeler à partir de scripts de navigateur, sans télécharger et 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 for JavaScriptGuide de référence de l'API V3
- AWSGuide de référence des SDK et des outils : contient des paramètres, des fonctionnalités et d'autres concepts fondamentaux communs aux SDK. AWS
- JavaScript Blog du développeur
- AWS JavaScript Forum
- JavaScript exemples dans le catalogue AWS de codes
- AWSRéférentiel d'exemples de code
- Chaîne Gitter
- Stack Overflow
- Questions relatives à Stack Overflow taggeDAWS -sdk-js
- GitHub
  - Source du SDK
  - Source de documentation

Premiers pas avec AWS SDK for JavaScript

AWS SDK for JavaScriptPermet 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 SDK JavaScript dans chacun de ces JavaScript environnements.

> **Note**
>
> Vous pouvez développer des applications Node.js et JavaScript des applications basées sur un navigateur à l'aide du SDK intégré JavaScript à l'IDE. AWS Cloud9 Pour un exemple d'utilisation AWS Cloud9 pour le développement de Node.js, consultezÀ utiliser AWS Cloud9 avec AWS SDK for JavaScript.

Rubriques

- Authentification du SDK avec AWS
- Commencez avec Node.js
- Commencez dans le navigateur

Authentification du SDK avec AWS

Vous devez définir la manière dont votre code est authentifié auprès d'AWS lors du développement avec les Services AWS. Vous pouvez configurer l'accès programmatique aux AWS ressources de différentes manières en fonction de l'environnement et de l'AWSaccès 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 AWS des SDK et des outils.

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 configurerAWS IAM Identity Center. Cette méthode consiste à installer le 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 des SDK et des outils :

- AWS CLI, que vous utilisez pour démarrer une session de portail d'accès AWS avant d'exécuter votre application.

- AWSconfigFichier 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, consultez Location of the shared files dans le manuel AWS SDKs and Tools Reference Guide.
- 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 section `sso-session` contient les paramètres permettant de lancer une session de portail d'accès AWS.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

La AWS SDK for JavaScript 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).

Démarrage d’une session de portail d'accès AWS

Avant d'exécuter une application qui y accèdeServices AWS, vous avez besoin d'une session de portail d'AWSaccès 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 d'accès AWS, exécutez la commande suivante dans 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 de portail d’accès AWS active et que vous exécutez `aws sso login`, il ne vous sera pas demandé de fournir des informations d'identification.
> Le processus de connexion peut vous demander d'autoriser AWS CLI à accéder à vos données. Comme AWS CLI il repose sur le SDK pour Python, les messages d'autorisation peuvent contenir des variantes du `botocore` nom.

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 avoir 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'accèsAWS. Vous pouvez utiliser un fournisseur d'identité pour vos utilisateurs humains afin de fournir un accès fédéré à des AWS

comptes en assumant des rôles, qui fournissent 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 informations d'identification AWS à court terme, consultez la section Informations d'identification de sécurité temporaires dans IAM dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur les autres fournisseurs d'informations d'identification AWS SDK for JavaScript V3, consultez la section Fournisseurs d'informations d'identification standardisés dans le Guide de référence AWSdes SDK et des outils.

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

Prérequis

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

- Configurez l'authentification de votre SDK. Pour plus d'informations, consultez Authentification du SDK avec AWS.
- Installez Node.js.

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

Pour configurer la structure des packages et installer les packages clients, procédez comme suit :

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 "readline/promises";

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

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

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

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

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

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

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

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

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

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

// Call a function if this file was run directly. This allows the file
// to be runnable without running on import.
```

```
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

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 SDK JavaScript dans le navigateur.

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

Pour d'autres exemples d'utilisation (V3) du 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'AWS Identity and Access Managementun 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 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 AWS Identity and Access Management (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 remplacer IDENTITY_POOL_ID et REGION dans.* Étape 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 AWS Management Console, puis ouvrez la console IAM à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le panneau 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 choisissez Joindre 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 Add permissions (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 AWS Management Console et ouvrez la console Amazon S3 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 :

   - Remplacez *`YOUR_IDENTITY_POOL_ID 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).
- Remplacez le *nom du compartiment par le nom du* compartiment dans lequel vous l'avez créé. Étape 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).

Configurer 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.

> **Note**
>
> Les développeurs de React Native devraient AWS Amplify l'utiliser pour créer de nouveaux projets sur AWS. Consultez les aws-sdk-react-nativearchives pour plus de détails.

Rubriques

- Prérequis
- Installez le SDK pour JavaScript
- Chargez le SDK pour JavaScript

Prérequis

Installez Node.js sur vos serveurs, s'il n'est pas déjà installé.

Rubriques

- Configuration d'un environnement AWS Node.js
- Navigateurs Web pris en charge

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) avec Node.js préinstallé. Créez ensuite une instance Amazon EC2 à l'aide de cette AMI. Lorsque vous créez votre instance Amazon EC2, 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 instance Amazon EC2 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 instance Amazon EC2.
- 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
- Créez un serveur d'applications Node.js à l'aide de AWS OpsWorks. Pour plus d'informations sur l'utilisation de Node.js avec AWS OpsWorks, consultez la section Création de votre première pile Node.js dans le Guide de AWS OpsWorks l'utilisateur.

Navigateurs Web pris en charge

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

Dans la version 3.183.0 ou ultérieure, le SDK JavaScript utilise les artefacts ES2020, 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+
Microsoft Edge	80,0 et plus
Apple Safari	14,1+
Samsung Internet	12,0 et plus

Dans la version 3.182.0 ou antérieure, le SDK JavaScript utilise les artefacts ES5, qui prennent 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 for JavaScript aide de npm, le gestionnaire de packages Node.js, entrez la commande suivante à l'invite de commande, où *SERVICE* est le nom d'un service, tel ques3.

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

Pour obtenir la liste complète des packages de AWS SDK for JavaScript service client, consultez le guide de référence des AWS SDK for 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 des AWS SDK et des outils contient également des paramètres, des fonctionnalités et d'autres concepts fondamentaux communs à de nombreux SDK. AWS

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
- Définissez la AWS région
- Définir les informations d'identification
- Considérations relatives à Node.js
- Considérations à propos des scripts du navigateur

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 for 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 des objets Amazon EC2 dans plusieurs AWS régions, créez un objet de service Amazon EC2 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 des `credentials` fichiers, consultez la section Fichiers de configuration et d'informations d'identification partagés dans le Guide de référence AWS des SDK et des outils.

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
- Définir les informations d'identification dans un navigateur Web

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 for 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 les SDK ont une série d'emplacements (ou de sources) qu'ils vérifient afin d'obtenir des informations d'identification valides à utiliser pour envoyer 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 AWS des SDK et des outils.

Le guide de référence AWS des SDK et des outils contient des informations sur les paramètres de configuration des SDK utilisés par tous les AWS SDK 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 for JavaScript vérifie les fournisseurs d'informations d'identification dans l'ordre indiqué dans le tableau suivant.

AWS SDK for JavaScrip t Méthode du fournisseur d'informations d'identification API Reference par ordre de priorité	Fournisseur (s) d'identification disponible (s)	AWS Guide de référence des SDK et des 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 for JavaScrip t Méthode du fournisseur d'informations d'identification API Reference par ordre de priorité	Fournisseur (s) d'identification disponible (s)	AWS Guide de référence des SDK et des 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 IAM Identity Center informations d'identification	Fournisseur d'identifiants IAM Identity Center
`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 for JavaScrip t Méthode du fournisseur d'informations d'identification API Reference par ordre de priorité	Fournisseur (s) d'identification disponible (s)	AWS Guide de référence des SDK et des outils
`fromInstanceMetada ta()`	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 AWS des SDK et des outils.

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 instance Amazon EC2, vous pouvez tirer parti des 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 instance Amazon EC2, 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 *`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;
};
```

Basculer les utilisateurs non authentifiés vers les 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 for JavaScript v3 modules.
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";
// Set the default credentials.
const creds = new 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 for 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. L'utilisation réussie de certaines API dépend de votre connaissance des modèles de codage Node.js courants, tels que l'importation et l'utilisation d'autres modules Node.js comme le module `File System (fs)`.

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 obtenir la 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 for 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, le SDK JavaScript réutilise les connexions TCP.

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.

Vous pouvez également désactiver le maintien de ces connexions en fonction du client par service, 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 { Agent } from "http";
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: 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 ce qui suit à l'invite de commande, où *PROXY* est 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 for 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. Pour un exemple d'utilisation de Webpack pour charger le SDK V3 JavaScript dans un navigateur, voir. Création d'une application pour envoyer des données à DynamoDB

Si vous travaillez avec le SDK en dehors d'un environnement qui applique le 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 for 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.

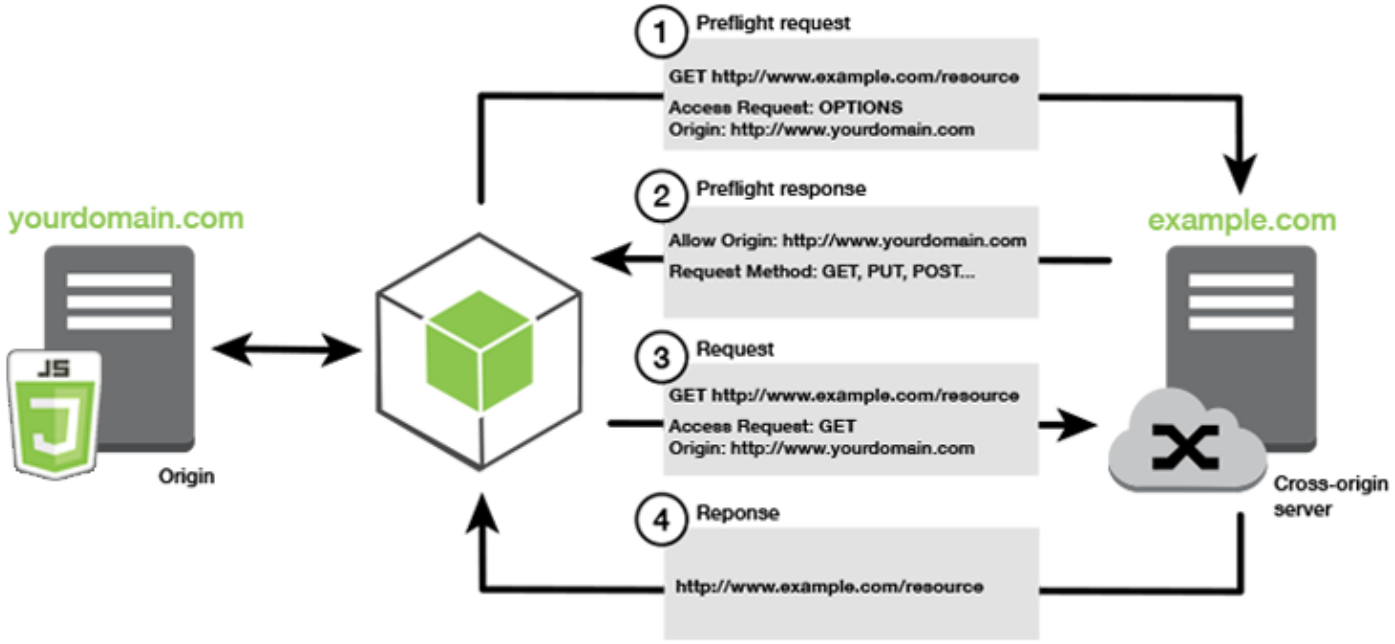
CORS détermine s'il convient d'autoriser le partage de ressources dans une demande d'origine croisée sur la base des éléments suivants :

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

Travaillez avec les AWS services du SDK pour JavaScript

La AWS SDK for 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 proposent des API de bas niveau pour utiliser les fonctionnalités et les ressources du service. Par exemple, les API Amazon DynamoDB sont disponibles par le biais de cette 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
- Appeler les services de manière asynchrone
- Création de demandes de service pour les clients
- Gérer les réponses des clients du service
- Travailler avec JSON
- SDK pour les exemples de JavaScript code

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.

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 rédigez 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
- Utiliser async/await
- JavaScript Promesses d'utilisation
- Utiliser une fonction de rappel anonyme

Gérer les appels asynchrones

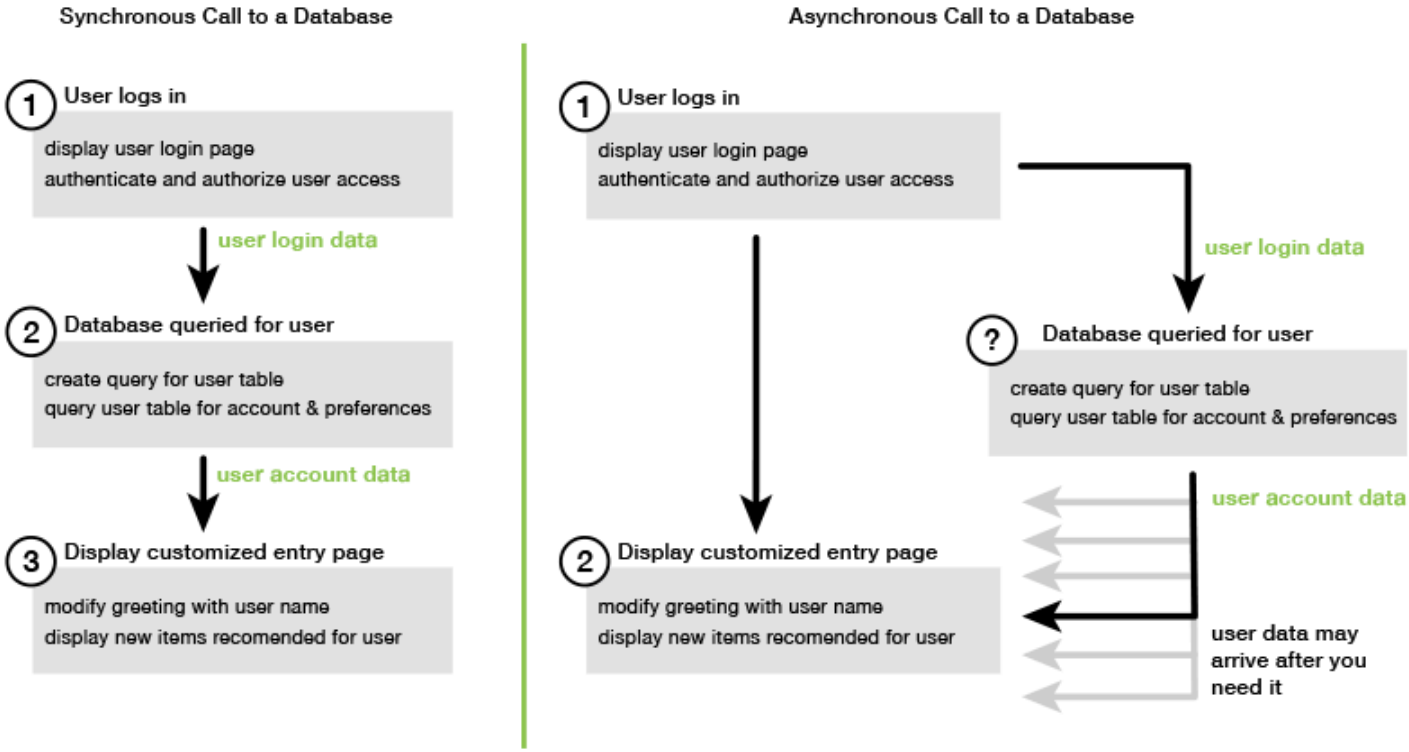
Par exemple, la page d'accueil d'un site e-commerce autorise le renvoi de la connexion des utilisateurs. Pour les clients qui se connectent, cela présente notamment l'avantage, après connexion, de disposer d'une personnalisation en fonction de leurs préférences. Pour cela :

1. Le client doit se connecter et être validé à l'aide de 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 for 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 for 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 des JavaScript promesses natives (ECMAScript 2015) dépend du JavaScript moteur et de la version dans lesquels votre code s'exécute. Pour déterminer le support des JavaScript promesses dans chaque environnement dans lequel votre code doit être exécuté, consultez le tableau de compatibilité ECMAScript 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.

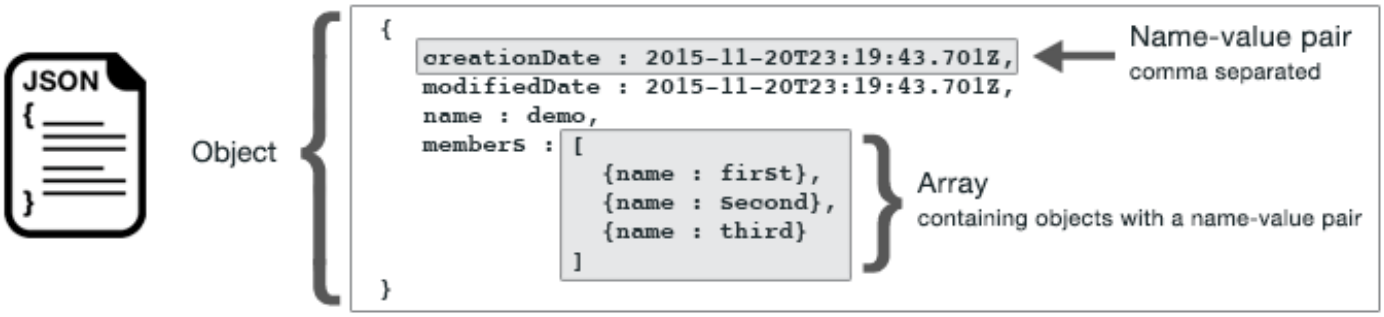
Informations sur les erreurs d'accès

Si une commande échoue, elle lance une exception. Vous pouvez gérer l'exception selon vos besoins.

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 for 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 };
};
```

SDK pour les exemples de JavaScript code

Les rubriques de cette section contiennent des exemples d'utilisation AWS SDK for JavaScript des API 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'import/export ECMAScript6.

- 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 Syntaxe ES6/CommonJS de conversion.

Rubriques

- JavaScript Syntaxe ES6/CommonJS
- Exemples d'Amazon DynamoDB
- Exemples AWS Elemental MediaConvert
- Exemples AWS Lambda
- Exemples d'Amazon Lex
- Exemples d'Amazon Polly
- Exemples d'Amazon Redshift
- Exemples d'Amazon Simple Email Service
- Exemples d'Amazon Simple Notification Service
- Exemples d'Amazon Transcribe
- Configuration de Node.js sur une instance Amazon EC2
- Création d'une application pour envoyer des données à DynamoDB
- Créez une application de transcription avec des utilisateurs authentifiés
- Invoquer Lambda avec API Gateway
- Création de flux de travail AWS sans serveur à l'aide de AWS SDK for JavaScript
- Création d'événements planifiés pour exécuter AWS Lambda des fonctions
- Création d'un chatbot Amazon Lex
- Création d'un exemple d'application de messagerie

JavaScript Syntaxe ES6/CommonJS

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

ES6 nécessite que vous utilisiez Node.js version 13.x ou supérieure. Pour télécharger et installer la version la plus récente d'EC2Lancer et installer la version la plus récente d'EC2Lancer Node.js Node.js Cependant, vous pouvez, selon vos préférences, convertir l'un de nos exemples en syntaxe CommonJS en suivant les étapes ci-dessous :

- `"type" : "module"`Supprimez-le de`package.json` l'environnement de votre projet.

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

  ```
  export {s3}
  ```

  À son équivalent CommonJS :

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

L'exemple suivant montre l'exemple de code permettant de créer un compartiment Amazon S3 à la fois dans ES6 et 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();
```

Exemples d'Amazon DynamoDB

Amazon DynamoDB est une base de données cloud NoSQL entièrement gérée qui prend en charge les modèles de magasin de documents et de magasins de valeurs clés. Vous créez des tables sans schéma pour les données sans avoir besoin d'allouer ni de gérer des serveurs de base de données dédiés.

L' JavaScript API pour DynamoDB est exposée par le biais `DynamoDB` des classes`DynamoDBStreams`,, `DynamoDB.DocumentClient` et client. Pour plus d'informations

sur l'utilisation des classes clientes DynamoDB, consultez Class : DynamoDB, Class : DynamoDBStreamset Class : DynamoDB utility dans le Guide de référence des API.

Rubriques

- Création et utilisation de tables dans DynamoDB
- Lecture et écriture d'un seul élément dans DynamoDB
- Lecture et écriture d'éléments par lots dans DynamoDB
- Interrogation et analyse d'une table DynamoDB
- Utilisation du client de documents DynamoDB

Création et utilisation de tables dans DynamoDB

Cet exemple de code Node.js présente :

- Comment créer et gérer les tables utilisées pour stocker et récupérer des données depuis DynamoDB.

Le scénario

Comme les autres systèmes de base de données, DynamoDB stocke les données dans des tables. Une table DynamoDB est un ensemble de données organisé en éléments analogues à des lignes. Pour stocker des données ou y accéder dans DynamoDB, vous devez créer des tables et les utiliser.

Dans cet exemple, vous utilisez une série de modules Node.js pour effectuer des opérations de base avec une table DynamoDB. Le code utilise le SDK pour JavaScript créer et utiliser des tables en utilisant les méthodes suivantes de la classe `DynamoDB` client :

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

Tâches prérequises

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

- Configurez l'environnement du projet pour exécuter ces exemples Node.js et installez les modules requis AWS SDK for JavaScript et tiers. Suivez les instructions figurant sur GitHub.
- Installez le SDK pour le client JavaScript DynamoDB. Pour plus d'informations, veuillez consulter Nouveautés de la version 3.
- 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 des SDK et des outils.

> **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 Syntaxe ES6/CommonJS.

> **Note**
>
> Pour plus d'informations sur les types de données utilisés dans ces exemples, consultez la section Types de données et règles de dénomination pris en charge dans Amazon DynamoDB.

Création d'une table

Créez un module Node.js nommé `create-table.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez `DynamoDB` un objet de service client. Créez un objet JSON contenant les paramètres requis pour créer une table, ce qui dans cet exemple inclut le nom et le type de données de chaque attribut, le schéma de clé, le nom de la table, ainsi que les unités de débit à allouer. Appelez la `CreateTableCommand` méthode de l'objet de service DynamoDB.

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

const client = new DynamoDBClient({});

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

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

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

```
node create-table.js
```

Cet exemple de code se trouve ici GitHub.

Répertorier vos tables

Créez un module Node.js nommé `list-tables.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez DynamoDB un objet de service client. Créez un objet JSON contenant les paramètres requis pour répertorier vos tables, ce qui dans cet exemple limite à 10 le nombre de tables répertoriées. Appelez la `ListTablesCommand` méthode de l'objet de service DynamoDB.

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

const client = new DynamoDBClient({});

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

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

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

```
node list-tables.js
```

Cet exemple de code se trouve ici GitHub.

Description d'une table

Créez un module Node.js nommé `describe-table.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez DynamoDB un objet de service client. Créez un objet JSON contenant les paramètres nécessaires pour décrire une `DescribeTableCommand` méthode de l'objet de service DynamoDB.

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
```

```
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

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

```
node describe-table.js
```

Cet exemple de code se trouve ici GitHub.

Suppression d'une table

Créez un module Node.js nommé `delete-table.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez `DynamoDB` un objet de service client. Créez un objet JSON contenant les paramètres requis pour décrire une table, ce qui dans cet exemple inclut le nom de la table fournie en tant que paramètre de ligne de commande. Appelez la `DeleteTableCommand` méthode de l'objet de service DynamoDB.

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

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

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

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

```
node delete-table.js
```

Cet exemple de code se trouve ici GitHub.

Lecture et écriture d'un seul élément dans DynamoDB

Cet exemple de code Node.js présente :

- Comment ajouter un élément dans une table DynamoDB
- Comment récupérer un élément dans une table DynamoDB
- Comment supprimer un élément dans une table DynamoDB

Le scénario

Dans cet exemple, vous utilisez une série de modules Node.js pour lire et écrire un élément dans une table DynamoDB en utilisant les méthodes suivantes de la `DynamoDB` classe client :

- PutItemCommand
- UpdateItemCommand
- GetItemCommand
- DeleteItemCommand

Tâches prérequises

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

- Configurez l'environnement du projet pour exécuter ces exemples Node.js et installez les modules requis AWS SDK for 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 des SDK et des outils.
- Créez une table DynamoDB dont vous pouvez accéder aux éléments. Pour plus d'informations sur la création d'une table DynamoDB, consultez. Création et utilisation de tables dans DynamoDB

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 Syntaxe ES6/CommonJS.

Note

Pour plus d'informations sur les types de données utilisés dans ces exemples, consultez la section Types de données et règles de dénomination pris en charge dans Amazon DynamoDB.

Écriture d'un élément

Créez un module Node.js nommé `put-item.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez `DynamoDB` un objet de service client. Créez un objet JSON contenant les paramètres requis pour ajouter un élément, ce qui dans cet exemple inclut le nom de la table et une carte qui détermine les attributs à définir et les valeurs de chaque attribut. Appelez la `PutItemCommand` méthode de l'objet de service client DynamoDB.

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

const client = new DynamoDBClient({});

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

```
    },
  });

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

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

```
node put-item.js
```

Cet exemple de code se trouve ici GitHub.

Mettre à jour un élément

Créez un module Node.js nommé `update-item.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez `DynamoDB` un objet de service client. Créez un objet JSON contenant les paramètres nécessaires pour ajouter un élément, qui, dans cet exemple, inclut le nom de la table, la clé à mettre à jour et l'expression de date qui mappe les nouveaux noms d'attributs, ainsi que les valeurs de chaque nouvel attribut. Appelez la `UpdateItemCommand` méthode de l'objet de service client DynamoDB.

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

const client = new DynamoDBClient({});

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

```
      ":chunks": { BOOL: "false" },
    },
    ReturnValues: "ALL_NEW",
  });

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

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

```
node update-item.js
```

Cet exemple de code se trouve ici GitHub.

Obtention d'un élément

Créez un module Node.js nommé `get-item.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez `DynamoDB` un objet de service client. Pour déterminer l'élément à obtenir, vous devez fournir la valeur de la clé primaire de cet élément dans la table. Par défaut, la méthode `GetItemCommand` renvoie toutes les valeurs d'attribut définies pour l'élément. Pour obtenir uniquement un sous-ensemble de toutes les valeurs d'attribut possible, spécifiez une expression de projection.

Créez un objet JSON contenant les paramètres requis pour obtenir un élément, ce qui dans cet exemple inclut le nom de la table, le nom et la valeur de la clé de l'élément que vous récupérez, ainsi qu'une expression de projection qui identifie l'attribut de l'élément à récupérer. Appelez la `GetItemCommand` méthode de l'objet de service client DynamoDB.

L'exemple de code suivant extrait un élément d'une table avec une clé primaire composée uniquement d'une clé de partition et non d'une clé de partition et d'une clé de tri. Si la table possède une clé primaire composée d'une clé de partition et d'une clé de tri, vous devez également spécifier le nom et l'attribut de la clé de tri.

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

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

```
export const main = async () => {
  const command = new GetItemCommand({
    TableName: "CafeTreats",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      TreatId: { N: "101" },
    },
  });

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

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

```
node get-item.js
```

Cet exemple de code se trouve ici GitHub.

Suppression d'un élément

Créez un module Node.js nommé `delete-item.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez DynamoDB un objet de service client. Créez un objet JSON contenant les paramètres requis pour supprimer un élément, ce qui dans cet exemple inclut le nom de la table, ainsi que le nom de clé et la valeur de l'élément que vous supprimez. Appelez la `DeleteItemCommand` méthode de l'objet de service client DynamoDB.

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

const client = new DynamoDBClient({});

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

```
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Name: { S: "Pumpkin Spice Latte" },
    },
  });

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

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

```
node delete-item.js
```

Cet exemple de code se trouve ici GitHub.

Lecture et écriture d'éléments par lots dans DynamoDB

Cet exemple de code Node.js présente :

- Comment lire et écrire des lots d'éléments dans une table DynamoDB

Le scénario

Dans cet exemple, vous utilisez une série de modules Node.js pour placer un lot d'éléments dans une table DynamoDB et lire un lot d'éléments. Le code utilise le SDK pour effectuer des opérations JavaScript de lecture et d'écriture par lots à l'aide des méthodes suivantes de la classe client DynamoDB :

- BatchGetItemCommand
- BatchWriteItemCommand

Tâches prérequises

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 for 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 des SDK et des outils.
- Créez une table DynamoDB dont vous pouvez accéder aux éléments. Pour plus d'informations sur la création d'une table DynamoDB, consultez. Création et utilisation de tables dans DynamoDB

> **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 Syntaxe ES6/CommonJS.

> **Note**
>
> Pour plus d'informations sur les types de données utilisés dans ces exemples, consultez la section Types de données et règles de dénomination pris en charge dans Amazon DynamoDB.

Lecture d'éléments par lots

Créez un module Node.js nommé `batch-get-item.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez `DynamoDB` un objet de service client. Créez un objet JSON contenant les paramètres requis pour obtenir un lot d'éléments, ce qui dans cet exemple inclut le nom d'une ou de plusieurs tables dans lesquelles lire les éléments, les valeurs des clés à lire dans chaque table et

l'expression de projection qui spécifie les attributs à renvoyer. Appelez la `BatchGetItemCommand` méthode de l'objet de service DynamoDB.

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

const client = new DynamoDBClient({});

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

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

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

```
node batch-get-item.js
```

Cet exemple de code se trouve ici GitHub.

Écrire des articles par lots

Créez un module Node.js nommé `batch-write-item.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez `DynamoDB` un objet de service client. Créez un objet JSON contenant les paramètres nécessaires pour obtenir un lot d'éléments, qui, dans cet exemple, inclut la table dans laquelle vous souhaitez écrire des éléments, les clés que vous souhaitez écrire pour chaque élément et les attributs ainsi que leurs valeurs. Appelez la `BatchWriteItemCommand` méthode de l'objet de service DynamoDB.

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

const client = new DynamoDBClient({});

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

```
          },
        },
        {
          PutRequest: {
            Item: {
              Name: { S: "Flora Ethiopia" },
              Process: { S: "Washed" },
              Flavors: { SS: ["Stone Fruit", "Toasted Almond", "Delicate"] },
            },
          },
        },
      ],
    },
  });

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

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

```
node batch-write-item.js
```

Cet exemple de code se trouve ici GitHub.

Interrogation et analyse d'une table DynamoDB

Cet exemple de code Node.js présente :

- Comment interroger et analyser une table DynamoDB à la recherche d'éléments.

Le scénario

L'interrogation recherche les éléments d'une table ou d'un index secondaire uniquement à l'aide des valeurs d'attribut de clé primaire. Vous devez fournir un nom de clé de partition et une valeur

à rechercher. Vous pouvez également indiquer un nom de clé de tri et une valeur, et utiliser un opérateur de comparaison pour affiner les résultats de recherche. L'analyse recherche les éléments en vérifiant chacun d'eux dans la table spécifiée.

Dans cet exemple, vous utilisez une série de modules Node.js pour identifier un ou plusieurs éléments que vous souhaitez récupérer dans une table DynamoDB. Le code utilise le SDK pour interroger et analyser des tables JavaScript à l'aide des méthodes suivantes de la classe client DynamoDB :

- QueryCommand
- ScanCommand

Tâches prérequises

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

- Configurez l'environnement du projet pour exécuter ces exemples Node.js et installez les modules requis AWS SDK for 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 des SDK et des outils.
- Créez une table DynamoDB dont vous pouvez accéder aux éléments. Pour plus d'informations sur la création d'une table DynamoDB, consultez. Création et utilisation de tables dans DynamoDB

> **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 Syntaxe ES6/CommonJS.

> **Note**
>
> Pour plus d'informations sur les types de données utilisés dans ces exemples, consultez la section Types de données et règles de dénomination pris en charge dans Amazon DynamoDB.

Interrogation d'une table

Créez un module Node.js nommé `query.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez `DynamoDB` un objet de service client. Créez un objet JSON contenant les paramètres requis pour interroger la table, ce qui dans cet exemple inclut le nom de la table, les valeurs `ExpressionAttributeValues` requises par la requête, une `KeyConditionExpression` qui utilise ces valeurs pour définir quels éléments sont renvoyés par la requête, ainsi que les noms des valeurs d'attribut à renvoyer pour chaque élément. Appelez la `QueryCommand` méthode de l'objet de service DynamoDB.

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

const client = new DynamoDBClient({});

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

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

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

```
node query.js
```

Cet exemple de code se trouve ici GitHub.

Analyse d'une table

Créez un module Node.js nommé `scan.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Pour accéder à DynamoDB, créez DynamoDB un objet de service client. Créez un objet JSON contenant les paramètres requis pour analyser les éléments de la table, ce qui dans cet exemple inclut le nom de la table, la liste des valeurs d'attribut à renvoyer pour chaque élément correspondant et une expression permettant de filtrer le jeu de résultats afin de rechercher les éléments contenant une expression spécifiée. Appelez la `ScanCommand` méthode de l'objet de service DynamoDB.

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

const client = new DynamoDBClient({});

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

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

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

```
node scan.js
```

Cet exemple de code se trouve ici GitHub.

Utilisation du client de documents DynamoDB

Cet exemple de code Node.js présente :

- Comment accéder à une table DynamoDB à l'aide des utilitaires DynamoDB

Scénario

Le client de documents DynamoDB simplifie l'utilisation des éléments en faisant abstraction de la notion de valeurs d'attribut. Cette abstraction annote les JavaScript types natifs fournis en tant que paramètres d'entrée et convertit les données de réponse annotées en types natifs JavaScript.

Pour plus d'informations sur le client de documents DynamoDB, consultez @aws -sdk/lib-dynamodb README on. GitHub Pour plus d'informations sur la programmation avec Amazon DynamoDB, consultez la section Programmation avec DynamoDB dans le manuel du développeur Amazon DynamoDB.

Dans cet exemple, vous utilisez une série de modules Node.js pour effectuer des opérations de base sur une table DynamoDB à l'aide des utilitaires DynamoDB. Le code utilise le SDK pour interroger et scanner des tables JavaScript à l'aide des méthodes suivantes de la classe DynamoDB Document Client :

- GetCommand
- PutCommand
- UpdateCommand
- QueryCommand
- DeleteCommand

Pour plus d'informations sur la configuration du client de documents DynamoDB, consultez @aws - sdk/lib-dynamodb.

Tâches prérequises

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

- Configurez l'environnement du projet pour exécuter ces exemples Node.js et installez les modules requis AWS SDK for 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 des SDK et des outils.
- Créez une table DynamoDB dont vous pouvez accéder aux éléments. Pour plus d'informations sur la création d'une table DynamoDB à l'aide du SDK JavaScript pour, consultez. Création et utilisation de tables dans DynamoDB Vous pouvez également utiliser la console DynamoDB pour créer une table.

> **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 Syntaxe ES6/CommonJS.

> **Note**
>
> Pour plus d'informations sur les types de données utilisés dans ces exemples, consultez la section Types de données et règles de dénomination pris en charge dans Amazon DynamoDB.

Obtention d'un élément à partir d'une table

Créez un module Node.js nommé `get.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Cela inclut `@aws-sdk/lib-dynamodb` un package de bibliothèque qui fournit des fonctionnalités de client de documents à`@aws-sdk/client-dynamodb`. Définissez ensuite la configuration comme indiqué ci-dessous pour le marshaling et le démarshaling (en tant que second paramètre facultatif) lors de la création du client de documents. Créez ensuite les clients. Créez maintenant un objet JSON contenant les paramètres nécessaires pour obtenir un élément de la table, qui, dans cet exemple, inclut le nom de la table, le nom de la clé de hachage de cette table et la valeur de la clé de hachage pour l'élément que vous souhaitez obtenir. Appelez la `GetCommand` méthode du client de documents DynamoDB.

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

```
node get.js
```

Cet exemple de code se trouve ici GitHub.

Positionnement d'un élément dans une table

Créez un module Node.js nommé `put.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Cela inclut `@aws-sdk/lib-dynamodb` un package de bibliothèque qui fournit des fonctionnalités de client de documents à`@aws-sdk/client-dynamodb`. Définissez ensuite la configuration comme indiqué ci-dessous pour le marshaling et le démarshaling (en tant que second paramètre facultatif) lors de la création du client de documents. Créez ensuite les clients. Créez un objet JSON contenant les paramètres nécessaires pour écrire un élément dans la table, qui, dans cet exemple, inclut le nom de la table et une description de l'élément à ajouter ou à mettre à jour qui inclut la clé de hachage et la valeur, ainsi que les noms et valeurs des attributs à définir sur l'élément. Appelez la `PutCommand` méthode du client de documents DynamoDB.

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

```
node put.js
```

Cet exemple de code se trouve ici GitHub.

Mise à jour d'un élément dans une table

Créez un module Node.js nommé `update.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Cela inclut `@aws-sdk/lib-dynamodb` un package de bibliothèque qui fournit des fonctionnalités de client de documents à`@aws-sdk/client-dynamodb`. Définissez ensuite la configuration comme indiqué ci-dessous pour le marshaling et le démarshaling (en tant que second paramètre facultatif) lors de la création du client de documents. Créez ensuite les clients. Créez un objet JSON contenant les paramètres nécessaires pour écrire un élément dans la table, qui, dans cet exemple, inclut le nom de la table, la clé de l'élément à mettre à jour, un ensemble définissant les attributs de l'élément à mettre à jour avec des jetons `UpdateExpressions` auxquels vous attribuez des valeurs dans les `ExpressionAttributeValues` paramètres. Appelez la méthode du client `UpdateCommand` de documents DynamoDB.

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

```
node update.js
```

Cet exemple de code se trouve ici GitHub.

Interrogation d'une table

Créez un module Node.js nommé `query.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Cela inclut `@aws-sdk/lib-dynamodb` un package de bibliothèque qui fournit des fonctionnalités de client de documents à`@aws-sdk/client-dynamodb`. Créez un objet JSON contenant les paramètres requis pour interroger la table, ce qui dans cet exemple inclut le nom de la table, les valeurs `ExpressionAttributeValues` requises par la requête et une `KeyConditionExpression` qui utilise ces valeurs pour définir quels éléments sont renvoyés par la requête. Appelez la `QueryCommand` méthode du client de documents DynamoDB.

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

```
node query.js
```

Cet exemple de code se trouve ici GitHub.

Suppression d'un élément d'une table

Créez un module Node.js nommé `delete.js`. Assurez-vous de configurer le SDK comme indiqué précédemment, notamment en installant les clients et les packages requis. Cela inclut `@aws-sdk/lib-dynamodb` un package de bibliothèque qui fournit des fonctionnalités de client de documents à`@aws-sdk/client-dynamodb`. Définissez ensuite la configuration comme indiqué ci-dessous pour le marshaling et le démarshaling (en tant que second paramètre facultatif) lors de la création du client de documents. Créez ensuite les clients. Pour accéder à DynamoDB, créez un objet. `DynamoDB` Créez un objet JSON contenant les paramètres nécessaires pour supprimer un élément de la table, qui, dans cet exemple, inclut le nom de la table ainsi que le nom et la valeur de la clé de hachage de l'élément que vous souhaitez supprimer. Appelez la `DeleteCommand` méthode du client de documents DynamoDB.

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

```
node delete.js
```

Cet exemple de code se trouve ici GitHub.

Exemples AWS Elemental MediaConvert

AWS Elemental MediaConvert est un service de transcodage vidéo basé sur des fichiers avec des fonctions de niveau diffuseur. 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 AWS Elemental MediaConvertGuide de l'utilisateur .

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

- Obtenir un point de terminaison spécifique à votre région pour MediaConvert
- Création et gestion de tâches de transcodage dans MediaConvert
- Utilisation de modèles de tâches dans MediaConvert

Obtenir un point de terminaison spécifique à votre région pour MediaConvert

Cet exemple de code Node.js présente :

- Comment récupérer votre point de terminaison spécifique à une région à partir de. MediaConvert

Le scénario

Dans cet exemple, vous utilisez un module Node.js pour appeler MediaConvert et récupérer votre point de terminaison spécifique à une région. Vous pouvez récupérer l'URL de votre point de terminaison à partir du point de terminaison par défaut du service et vous n'avez donc pas encore besoin de votre point de terminaison spécifique à une région. Le code utilise le SDK JavaScript pour récupérer ce point de terminaison en utilisant cette méthode de la classe MediaConvert client :

- `DescribeEndpointsCommand`

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 for 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 des SDK et des outils.
- 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 MediaConvertutilisateur.

> **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 Syntaxe ES6/CommonJS.

Obtenir l'URL de votre point de terminaison

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

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the AWS Region.
const REGION = "REGION";
//Set the MediaConvert Service Object
const emcClientGet = new MediaConvertClient({ region: REGION });
export { emcClientGet };
```

Cet exemple de code se trouve ici GitHub.

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

Créez un objet pour transmettre les paramètres de requête vides pour la `DescribeEndpointsCommand` méthode de la classe MediaConvert client. Ensuite, appelez la méthode `DescribeEndpointsCommand`.

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

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

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

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

```
node emc_getendpoint.js
```

Cet exemple de code se trouve ici GitHub.

Création et gestion de tâches de transcodage dans MediaConvert

Cet exemple de code Node.js présente :

- Comment spécifier le point de terminaison spécifique à la région à utiliser avec. MediaConvert

- 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 for 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 des SDK et des outils.
- 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 MediaConvertutilisateur.

> **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 Syntaxe ES6/CommonJS.

Configuration du kit SDK

Configurez le SDK comme indiqué précédemment, notamment en téléchargeant les clients et les packages requis. Dans la mesure où chaque compte MediaConvert utilise des points de terminaison personnalisés, vous devez également configurer la classe de `MediaConvert` client pour qu'elle utilise le point de terminaison spécifique à votre région. Pour ce faire, vous définissez le paramètre `endpoint` sur `mediaconvert(endpoint)`.

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

Définition d'une tâche de transcodage simple

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`emcClient.js`. Copiez-collez le code ci-dessous pour créer l'objet MediaConvert client. Remplacez *`REGION`* par votre AWS région. Remplacez *`ENDPOINT`* par le point de terminaison de votre MediaConvert compte, comme vous pouvez le faire sur la page Compte de la MediaConvert console.

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

```
export { emcClient };
```

Cet exemple de code se trouve ici GitHub.

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 MediaConvertconsole 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**
>
> *Remplacez JOB_QUEUE_ARN par la file d'attente des MediaConvert tâches, IAM_ROLE_ARN par le nom de ressource Amazon (ARN) du rôle IAM, OUTPUT_BUCKET_NAME par le nom du bucket de destination, par exemple « s3://OUTPUT_BUCKET_NAME/ », et INPUT_BUCKET_AND_NAME par le bucket d'entrée et le nom de 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 `libs` répertoire et créez un module Node.js avec le nom du fichier`emcClient.js`. Copiez-collez le code ci-dessous pour créer l'objet MediaConvert client. Remplacez *`REGION`* par votre AWS région. Remplacez *`ENDPOINT`* par le point de terminaison de votre MediaConvert compte, comme vous pouvez le faire sur la page Compte de la MediaConvert console.

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

Cet exemple de code se trouve ici GitHub.

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.

Répertorier les travaux de transcodage récents

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`emcClient.js`. Copiez-collez le code ci-dessous pour créer l'objet MediaConvert client. Remplacez *`REGION`* par votre AWS région. Remplacez *`ENDPOINT`* par le point de terminaison de votre MediaConvert compte, comme vous pouvez le faire sur la page Compte de la MediaConvert console.

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

Cet exemple de code se trouve ici GitHub.

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 du nom de ressource Amazon (ARN) de la file d'attente de tâches

à vérifier, ainsi que le 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**
>
> Remplacez *`QUEUE_ARN`* par le nom de ressource Amazon (ARN) de la file 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 :

- Procédure de création des modèles de tâche AWS Elemental MediaConvert.
- 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 for 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 des SDK et des outils.

- 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 MediaConvertutilisateur.

> **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 Syntaxe ES6/CommonJS.

Création d'un modèle de tâche

Créez un `libs` répertoire et créez un module Node.js avec le nom du fichier`emcClient.js`. Copiez-collez le code ci-dessous pour créer l'objet MediaConvert client. Remplacez *`REGION`* par votre AWS région. Remplacez *`ENDPOINT`* par le point de terminaison de votre MediaConvert compte, comme vous pouvez le faire sur la page Compte de la MediaConvert console.

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

Cet exemple de code se trouve ici GitHub.

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 *`JOB_QUEUE_ARN`* par le nom de ressource Amazon (ARN) de la file d'attente de 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 `libs` répertoire et créez un module Node.js avec le nom du fichier`emcClient.js`. Copiez-collez le code ci-dessous pour créer l'objet MediaConvert client. Remplacez *`REGION`* par votre AWS région. Remplacez *`ENDPOINT`* par le point de terminaison de votre MediaConvert compte, comme vous pouvez le faire sur la page Compte de la MediaConvert console.

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

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

Cet exemple de code se trouve ici GitHub.

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**
>
> *Remplacez JOB_QUEUE_ARN par le nom de ressource Amazon (ARN) de la file d'attente de tâches à vérifier, KEY_PAIR_NAME par, TEMPLATE_NAME par, ROLE_ARN par le nom de ressource Amazon (ARN) du rôle et INPUT_BUCKET_AND_NAME par le compartiment d'entrée et le nom de 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 `libs` répertoire et créez un module Node.js avec le nom du fichier`emcClient.js`. Copiez-collez le code ci-dessous pour créer l'objet MediaConvert client. Remplacez *`REGION`* par votre AWS région. Remplacez *`ENDPOINT`* par le point de terminaison de votre MediaConvert compte, comme vous pouvez le faire sur la page Compte de la MediaConvert console.

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

Cet exemple de code se trouve ici GitHub.

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 `libs` répertoire et créez un module Node.js avec le nom du fichier`emcClient.js`. Copiez-collez le code ci-dessous pour créer l'objet MediaConvert client. Remplacez *`REGION`* par votre AWS région. Remplacez *`ENDPOINT`* par le point de terminaison de votre MediaConvert compte, comme vous pouvez le faire sur la page Compte de la MediaConvert console.

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

Cet exemple de code se trouve ici GitHub.

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.

Exemples AWS Lambda

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

L'JavaScriptAPI 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 for JavaScript v3 :

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

Exemples d'Amazon Lex

Amazon Lex est un AWS service qui permet de créer des interfaces conversationnelles dans des applications à l'aide de la voix et du texte.

L'JavaScriptAPI pour 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 des SDK et des outils.
- Créez une enquête de rôle d'utilisateur Amazon Cognito non authentifiée AWS Identity and Access Management (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 AWS 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

```
{
  "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 AWS CloudFormation

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

Pour créer la AWS CloudFormation pile :

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

> **Note**
>
> Le AWS CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour en savoir plus sur le AWS CDK, veuillez consulter le Guide du développeur AWS Cloud Development Kit (AWS CDK).

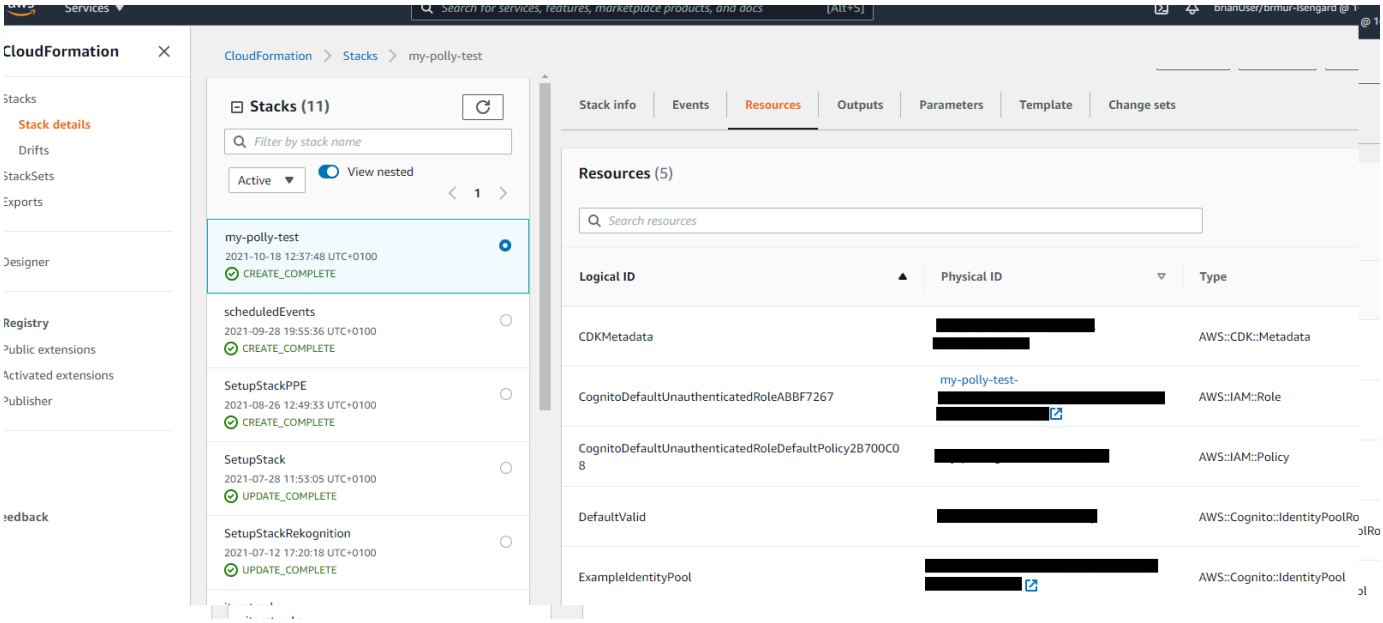
3. Exécutez la commande suivante depuis la ligne de commande, en remplaçant *`STACK_NAME par un nom`* unique pour la pile.

> **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 AWS CloudFormation l'utilisateur.

4. Accédez à la console AWS 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 la *`REGION`* et le *`BUCKET_NAME`*. Pour accéder à Amazon Polly, créez un objet de service `Polly` client. Remplacez *`« IDENTITY_POOL_ID »`* par celui `IdentityPoolId` 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.

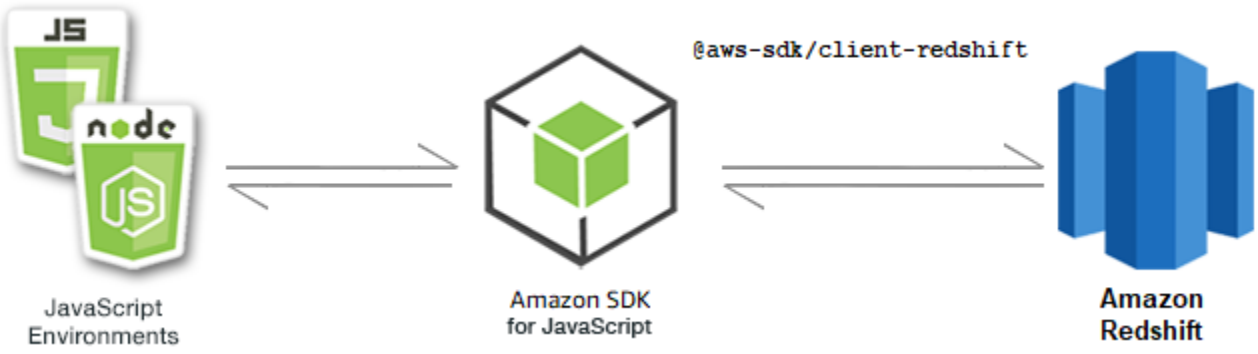
Appelez la `StartSpeechSynthesisCommand` méthode de l'objet du service client Amazon Polly, synthétisez le message vocal et chargez-le dans le compartiment Amazon S3.

```
const { StartSpeechSynthesisTaskCommand } = require("@aws-sdk/client-polly");
const { pollyClient } = require("./libs/pollyClient.js");

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

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

Cet exemple de code se trouve ici sur GitHub.

Exemples d'Amazon Redshift

Amazon Redshift est un service d'entreposage de données entièrement géré dans le cloud. Un entrepôt de 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 par le biais de 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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/CommonJS

Création d'un cluster Amazon Redshift

Cet exemple montre comment créer un cluster Amazon Redshift à l'aide du. AWS SDK for JavaScript Pour plus d'informations, consultez CreateCluster.

> **Important**
>
> Le cluster que vous êtes sur le point de créer est actif (et ne s'exécute 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*, spécifiez le type de nœud à provisionner, tel que « dc2.large », par exemple. *MASTER_USERNAME* et *MASTER_USER_PASSWORD* sont les informations 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.

Modification d'un cluster Amazon Redshift

Cet exemple montre comment modifier le mot de passe de l'utilisateur principal d'un cluster Amazon Redshift à l'aide du. AWS SDK for 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**
>
> Remplacez *CLUSTER_NAME 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 for 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 for 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 « false », 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();
```

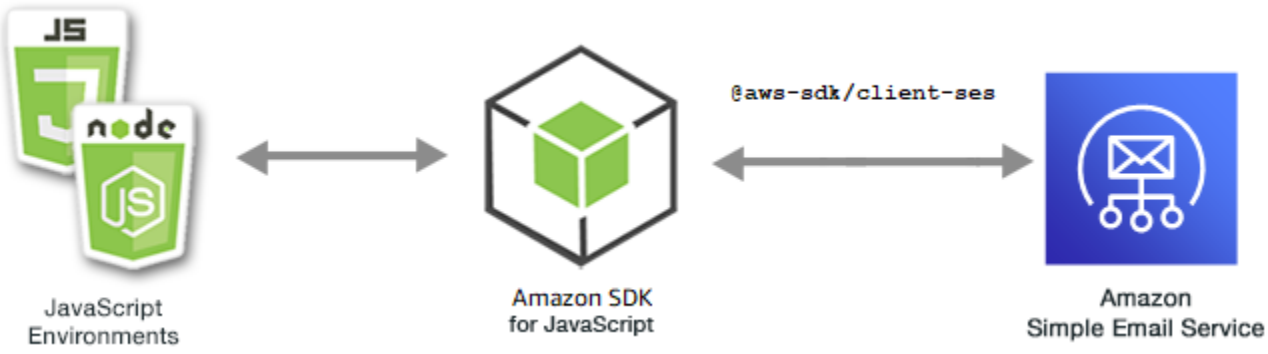
Pour exécuter l'exemple, entrez ce qui suit à l'invite de commande.

```
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
- Envoyer un e-mail 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 AWS Identity and Access Management (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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/ 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 *`IDENTITY_TYPE`* par le type d'identité, qui peut être « » ou EmailAddress « 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 *ADDRESS@DOMAIN.EXT* 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.

Vérifier 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 *`AMI_ID`* par l'ID de l'Amazon Machine Image (AMI) à exécuter, et *`KEY_PAIR_NAME`* de la paire de clés à attribuer à l'ID de l'AMI.

```
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_TYPE`* par le type d'identité à supprimer et *`IDENTITY_NAME par le nom`* 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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/ 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.

> **Note**
>
> Remplacez *`ITEMS_COUNT`* par le nombre maximum de modèles à renvoyer. La valeur doit être au minimum de 1 et au maximum de 10.

```
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 (err) {
    console.log("Failed to get email 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_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**
>
> 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 nouveau modèle, *HTML_CONTENT par le contenu* balisé HTML de l'e-mail, SUBJECT par l'*objet* de l'e-mail et *TEXT_CONTENT* par le texte 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**
>
> Remplacez *TEMPLATE_NAME* par le nom du modèle, *HTML_CONTENT par le contenu* balisé HTML de l'e-mail, SUBJECT par l'*objet* de l'e-mail et *TEXT_CONTENT* par le texte 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.

Envoyer un e-mail 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 de l'e-mail : 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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/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 *`RECEIVER_ADDRESS`* par l'adresse à laquelle envoyer l'e-mail, et *`SENDER_ADDRESS par l'adresse`* e-mail à partir de laquelle envoyer l'e-mail.

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

```
const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
```

```
    return await sesClient.send(sendEmailCommand);
  } catch (e) {
    console.error("Failed to send email.");
    return e;
  }
};
```

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.

Envoyer 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, *RECEIVER_ADDRESS* par l'adresse à laquelle envoyer l'e-mail, *SENDER_ADDRESS 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 (err) {
    console.log("Failed to send template email", err);
    return err;
  }
};
```

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 *`RECEIVER_ADDRESSES`* par l'adresse à laquelle envoyer l'e-mail, et *`SENDER_ADDRESS par l'adresse`* e-mail à partir de laquelle envoyer l'e-mail.

```
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 (err) {
    console.log("Failed to send bulk template email", err);
```

```
    return err;
  }
};
```

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 d'Amazon Simple Notification Service

Amazon Simple Notification Service (Amazon SNS) est un service web qui coordonne et gère la mise à disposition ou l'envoi de messages à des clients ou à des points de terminaison 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, SMSAWS 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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/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 (ARN).

```
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**
>
> Remplacez *`TOPIC_ARN`* 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 de la rubrique.

```
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 *`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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/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 d'informations sur les attributs SMS disponibles, consultez SetSMSAttributes.

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**
>
> Remplacez *`MESSAGE_TEXT 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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/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**
>
> Remplacez *TOPIC_ARN* par le nom de ressource Amazon (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 montrent 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**
>
> Remplacez *`TOPIC_ARN`* par le nom de ressource Amazon (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.

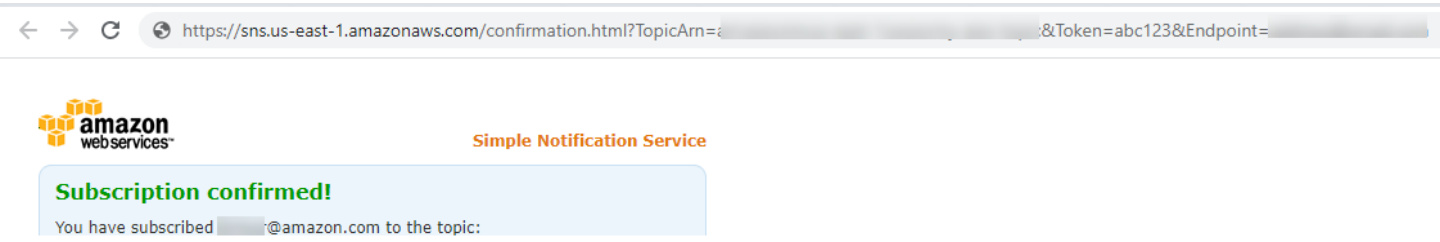
```
import { SNSClient } from "@aws-sdk/client-sns";

// 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**
>
> Remplacez *TOPIC_ARN* par le nom de ressource Amazon (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 ou. `TRUE` `FALSE`

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

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

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 les 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-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 *TOPIC_ARN* par le nom de ressource Amazon (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, 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 *`TOPIC_ARN`* par le nom de ressource Amazon (ARN) du sujet, et *`LAMBDA_FUNCTION_ARN`* par le nom de ressource Amazon (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**
>
> Remplacez *`TOPIC_SUBSCRIPTION_ARN par le nom de ressource Amazon (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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/ 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 SetSMSAttributes 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 SetSMSAttributes 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 apparaît sous la forme +1001XXX5550100.

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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/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 for JavaScript Pour plus d'informations, consultez 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.

```
const { TranscribeClient } = require("@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**
>
> Remplacez *`MEDICAL_JOB_NAME par le nom`* de la tâche de transcription. Pour *`OUTPUT_BUCKET_NAME, spécifiez`* le compartiment Amazon S3 dans lequel la sortie est enregistrée. Pour *`JOB_TYPE`*, spécifiez les types de tâches. Pour *`SOURCE_LOCATION`*, spécifiez l'emplacement du fichier source. Pour *`SOURCE_FILE_LOCATION`*, spécifiez 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 for 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.

```
const { TranscribeClient } = require("@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**
>
> Remplacez *`KEY_WORD 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 for 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 for 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 des SDK et des outils.

> **Important**
>
> Ces exemples montrent comment importer/exporter des objets et des commandes du service client à l'aide d'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 Syntaxe ES6/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 for 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**
>
> Remplacez *`MEDICAL_JOB_NAME par le nom`* de la tâche de transcription médicale. Pour *`OUTPUT_BUCKET_NAME, spécifiez`* le compartiment Amazon S3 dans lequel la sortie est enregistrée. Pour *`JOB_TYPE`*, spécifiez les types de tâches. Pour *`SOURCE_LOCATION`*, spécifiez l'emplacement du fichier source. Pour *`SOURCE_FILE_LOCATION`*, spécifiez 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 for JavaScript Pour plus d'informations, consultez 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.

```
const { TranscribeClient } = require("@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**
>
> Remplacez *KEYWORD* 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 for 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 instance Amazon EC2

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.

Prérequis

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 l'utilisateur Amazon EC2 pour les instances Linux.

> **Important**
>
> Utilisez l'Amazon Machine Image (AMI) Amazon Linux 2023 lors du lancement d'une nouvelle instance Amazon EC2.

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'utilisateur Amazon EC2 pour les instances Linux.

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 s'applique uniquement à la session Amazon EC2 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'instance Amazon EC2 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 instance Amazon EC2, vous pouvez créer une Amazon Machine Image (AMI) à partir de cette instance. La création d'une AMI facilite le provisionnement de plusieurs instances Amazon EC2 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 l'utilisateur Amazon EC2 pour les instances Linux.

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.

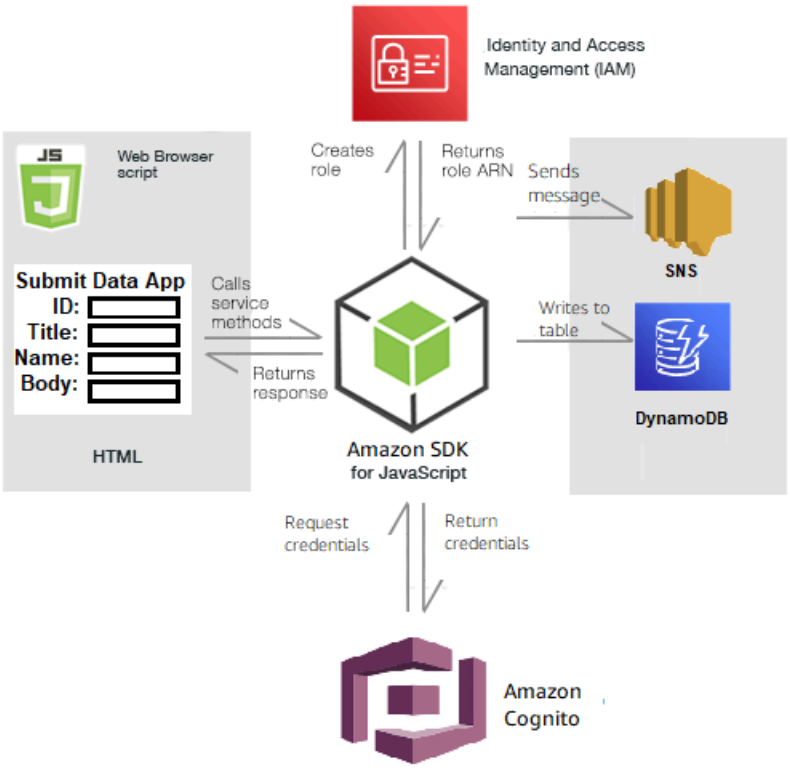
Création d'une application pour envoyer des données à DynamoDB

Ce didacticiel Node.js multiservice explique comment créer une application permettant aux utilisateurs de soumettre des données à une table Amazon DynamoDB. Cette application utilise les services suivants :

- AWS Identity and Access Management(IAM) et Amazon Cognito pour les autorisations et autorisations.
- Amazon DynamoDB (DynamoDB) pour créer et mettre à jour les tables.
- Amazon Simple Notification Service (Amazon SNS) pour avertir l'administrateur de l'application lorsqu'un utilisateur met à jour le tableau.

Le scénario

Dans ce didacticiel, une page HTML fournit une application basée sur un navigateur pour envoyer des données à une table Amazon DynamoDB. L'application utilise Amazon SNS pour avertir l'administrateur de l'application lorsqu'un utilisateur met à jour le tableau.



Pour créer l'application :

1. Prérequis
2. Allocation des ressources
3. Créez le code HTML
4. Créez le script du navigateur
5. Étapes suivantes

Prérequis

Effectuez les tâches préalables suivantes :

- Configurez l'environnement du projet pour exécuter ces TypeScript exemples de nœuds et installez les modules requis AWS SDK for 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 des SDK et des outils.

Créez les AWS ressources

Cette application nécessite les ressources suivantes :

- AWS Identity and Access Management(IAM) Rôle d'utilisateur Amazon Cognito non authentifié avec les autorisations suivantes :
  - sns:Publish
  - dynamodb : PutItem
- Une table DynamoDB.

Vous pouvez créer ces ressources manuellement dans la AWS console, mais nous vous recommandons de les approvisionner en suivant les AWS CloudFormation instructions de ce didacticiel.

Créez les AWS ressources en utilisant AWS CloudFormation

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

Pour créer la AWS CloudFormation pile à l'aide de AWS CLI :

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

   > **Note**
   >
   > Le AWS CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour en savoir plus sur le AWS CDK, veuillez consulter le Guide du développeur AWS Cloud Development Kit (AWS CDK).

3. Exécutez la commande suivante depuis la ligne de commande, en remplaçant *`STACK_NAME par un nom`* unique pour la pile et REGION dans votre *`région`*. AWS

   > **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 --region REGION
   ```

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

   Pour afficher les ressources créées, ouvrez AWS CloudFormation dans la console AWS de gestion, choisissez la pile, puis sélectionnez l'onglet Ressources.

4. Lorsque la pile est créée, utilisez le AWS SDK for JavaScript pour remplir la table DynamoDB, comme décrit dans. Remplissage 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. Remplacez *`REGION`* par votre AWS région et remplacez *`IDENTITY_POOL_ID par un identifiant de pool d'`*identités Amazon Cognito. Cela crée l'objet client DynamoDB.

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

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

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

export { dynamoClient };
```

Ce code est disponible ici GitHub.

Ensuite, créez un `dynamoAppHelperFiles` dossier dans le dossier de votre projet, créez-y un fichier `update-table.js` et GitHubcopiez-y le contenu.

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

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

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

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

Exécutez la commande suivante depuis la ligne de commande.

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

Ce code est disponible ici GitHub.

Création d'une page frontale pour l'application

Vous créez ici la page du navigateur HTML frontal pour l'application.

Créez un DynamoDBApp répertoire, créez un fichier nommé `index.html` et copiez le code à partir de maintenant GitHub. L'`script`élément ajoute le `main.js` fichier, qui contient tous les éléments requis JavaScript pour l'exemple. Vous créerez le `main.js` fichier ultérieurement dans ce didacticiel. Le code restant `index.html` crée la page du navigateur qui capture les données saisies par les utilisateurs.

Cet exemple de code se trouve ici GitHub.

Créez le script du navigateur

Créez d'abord les objets client de service requis pour l'exemple. Créez un `libs` répertoire`snsClient.js`, créez et collez-y le code ci-dessous. Remplacez *`REGION`* et *`IDENTITY_POOL_ID`* dans chacun d'eux.

> **Note**
>
> Utilisez l'ID du pool d'identités Amazon Cognito dans lequel vous avez créé. Créez les AWS ressources

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

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

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

export { snsClient };
```

Ce code est disponible ici GitHub. .

Pour créer le script de navigateur correspondant à cet exemple, dans un dossier nomméDynamoDBApp, créez un module Node.js portant le nom de fichier `add_data.js` et collez-y le code ci-dessous. La `submitData` fonction envoie des données à une table DynamoDB et envoie un SMS à l'administrateur de l'application via Amazon SNS.

Dans la `submitData` fonction, déclarez les variables pour le numéro de téléphone cible, les valeurs saisies sur l'interface de l'application et pour le nom du compartiment Amazon S3. Créez ensuite un objet de paramètres pour ajouter un élément au tableau. Si aucune des valeurs n'est vide, `submitData` ajoute l'élément au tableau et envoie le message. N'oubliez pas de mettre la fonction à la disposition du navigateur, avec`window.submitData = submitData`.

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

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

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

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

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

Cet exemple de code se trouve ici GitHub.

Enfin, exécutez la commande suivante à l'invite de commande JavaScript pour regrouper le fichier correspondant à cet exemple dans un fichier nommé `main.js` :

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

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

Pour exécuter l'application, `index.html` ouvrez-la dans votre navigateur.

Supprimer les ressources

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 AWS CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

1. Ouvrez le AWS CloudFormationdans la console AWS de gestion.
2. Ouvrez la page Stacks et sélectionnez la pile.
3. Sélectionnez Delete (Supprimer).

Pour d'autres exemples AWS interservices, consultez les exemples AWS SDK for JavaScriptinterservices.

Créez une application de transcription avec des utilisateurs authentifiés

Dans ce didacticiel, vous allez découvrir comment :

- Mettez en œuvre l'authentification à l'aide d'un pool d'identités Amazon Cognito pour accepter les utilisateurs fédérés avec un groupe d'utilisateurs Amazon Cognito.
- Utilisez Amazon Transcribe pour transcrire et afficher les enregistrements vocaux dans le navigateur.

Le scénario

L'application permet aux utilisateurs de s'inscrire avec une adresse e-mail et un nom d'utilisateur uniques. Après confirmation de leur e-mail, ils peuvent enregistrer des messages vocaux qui sont automatiquement transcrits et affichés dans l'application.

Comment ça marche

L'application utilise deux compartiments Amazon S3, l'un pour héberger le code de l'application et l'autre pour stocker les transcriptions. L'application utilise un groupe d'utilisateurs Amazon Cognito pour authentifier vos utilisateurs. Les utilisateurs authentifiés disposent des autorisations IAM pour accéder aux services requis. AWS

La première fois qu'un utilisateur enregistre un message vocal, Amazon S3 crée un dossier unique avec le nom de l'utilisateur dans le compartiment Amazon S3 pour stocker les transcriptions. Amazon Transcribe transcribe le message vocal en texte et l'enregistre au format JSON dans le dossier de l'utilisateur. Lorsque l'utilisateur actualise l'application, ses transcriptions s'affichent et peuvent être téléchargées ou supprimées.

Le didacticiel devrait prendre environ 30 minutes.

Étapes

Pour créer l'application, procédez comme suit :

1. Prérequis
2. Créez les AWS ressources
3. Créez le code HTML
4. Préparez le script du navigateur
5. Exécutez l'application
6. Supprimer les ressources

Prérequis

- Configurez l'environnement du projet pour exécuter ces JavaScript exemples de nœuds et installez les modules requis AWS SDK for 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 des SDK et des outils.

> **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 Syntaxe ES6/CommonJS.

Créez les AWS ressources

Cette section décrit comment provisionner AWS des ressources pour cette application à l'aide duAWS Cloud Development Kit (AWS CDK).

> **Note**
>
> AWS CDKIl s'agit d'un framework de développement logiciel qui vous permet de définir les ressources des applications cloud. Pour plus d'informations, consultez le Guide du développeur AWS Cloud Development Kit (AWS CDK).

Pour créer des ressources pour l'application, utilisez le modèle ci-dessous GitHub pour créer une AWS CDK pile à l'aide de la console de gestion des services AWS Web ou du AWS CLI. Pour obtenir des instructions sur la façon de modifier la pile ou de supprimer la pile et ses ressources associées une fois le didacticiel terminé, cliquez ici GitHub.

> **Note**
>
> 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.

La pile résultante provisionne automatiquement les ressources suivantes.

- Un pool d'identités Amazon Cognito doté d'un rôle d'utilisateur authentifié.
- Une politique IAM avec des autorisations pour Amazon S3 et Amazon Transcribe est associée au rôle d'utilisateur authentifié.
- Un groupe d'utilisateurs Amazon Cognito qui permet aux utilisateurs de s'inscrire et de se connecter à l'application.
- Un compartiment Amazon S3 pour héberger les fichiers de l'application.
- Un compartiment Amazon S3 pour stocker les transcriptions.

> **Important**
>
> Ce compartiment Amazon S3 autorise l'accès public READ (LIST), ce qui permet à quiconque de répertorier les objets contenus dans le compartiment et d'utiliser ces informations à mauvais escient. Si vous ne supprimez pas ce compartiment Amazon S3 immédiatement après avoir terminé le didacticiel, nous vous recommandons vivement de suivre les meilleures pratiques de sécurité d'Amazon S3 décrites dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Créez le code HTML

Créez un `index.html` fichier, copiez-collez le contenu ci-dessous dans celui-ci. La page comporte un panneau de boutons pour enregistrer les messages vocaux et un tableau affichant les messages précédemment transcrits de l'utilisateur actuel. La balise de script située à la fin de l'`body`élément invoque le`main.js`, qui contient tous les scripts de navigateur de l'application. Vous créez le fichier main.js à l'aide de Webpack, comme décrit dans la section suivante de ce didacticiel.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>title</title>
    <link rel="stylesheet" type="text/css" href="recorder.css">
    <style>
        table, td {
            border: 1px solid black;
        }
    </style>
</head>
<body>
<h2>Record</h2>
<p>
    <button id="record" onclick="startRecord()"></button>
    <button id="stopRecord" disabled onclick="stopRecord()">Stop</button>
<p id="demo" style="visibility: hidden;"></p>
</p>
<p>
    <audio id="recordedAudio"></audio>
</p>

<h2>My transcriptions</h2>
<table id="myTable1" style ="width:678px;">
</table>
<table id="myTable" style ="width:678px;">
        <tr>
    <td style = "font-weight:bold">Time created</td>
    <td style = "font-weight:bold">Transcription</td>
    <td style = "font-weight:bold">Download</td>
    <td style = "font-weight:bold">Delete</td>
       </tr>
</table>
```

```
<script type="text/javascript" src="./main.js"></script>
</body>

</html>
```

Cet exemple de code est disponible ici GitHub.

Préparez le script du navigateur

Il existe trois fichiers,`index.html`, et `recorder.jshelper.js`, que vous devez regrouper en un seul à `main.js` l'aide de Webpack. Cette section décrit en détail uniquement les fonctions `index.js` qui utilisent le SDK pour JavaScript, qui est disponible ici. GitHub

> **Note**
>
> `recorder.js`et `helper.js` sont obligatoires mais, comme ils ne contiennent pas de code Node.js, ils sont expliqués dans les commentaires en ligne ici et ici respectivement. GitHub

Définissez d'abord les paramètres. `COGNITO_ID`Est le point de terminaison du groupe d'utilisateurs Amazon Cognito que vous avez créé dans le Créez les AWS ressources sujet de ce didacticiel. Il est formaté. `cognito-idp.`*`AWS_REGION`*`.amazonaws.com/`*`USER_POOL_ID`* L'identifiant du groupe d'utilisateurs est *`ID_TOKEN`* dans le jeton AWS d'identification, qui est supprimé de l'URL de l'application par la `getToken` fonction du fichier « helper.js ». Ce jeton est transmis à la `loginData` variable, qui fournit des identifiants de connexion aux objets clients Amazon Transcribe et Amazon S3. Remplacez *« REGION »* par la AWS région, et *« BUCKET »* par Remplacez *« IDENTITY_POOL_ID »* par celui de la page d'exemple **`IdentityPoolId`** du pool d'identités Amazon Cognito que vous avez créé pour cet exemple. Ceci est également transmis à chaque objet client.

```
// Import the required AWS SDK clients and commands for Node.js
import "./helper.js";
import "./recorder.js";
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import {
  CognitoIdentityProviderClient,
  GetUserCommand,
} from "@aws-sdk/client-cognito-identity-provider";
```

```
import { S3RequestPresigner } from "@aws-sdk/s3-request-presigner";
import { createRequest } from "@aws-sdk/util-create-request";
import { formatUrl } from "@aws-sdk/util-format-url";
import {
  TranscribeClient,
  StartTranscriptionJobCommand,
} from "@aws-sdk/client-transcribe";
import {
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
  ListObjectsCommand,
  DeleteObjectCommand,
} from "@aws-sdk/client-s3";
import fetch from "node-fetch";

// Set the parameters.
// 'COGINTO_ID' has the format 'cognito-idp.eu-west-1.amazonaws.com/COGNITO_ID'.
let COGNITO_ID = "COGNITO_ID";
// Get the Amazon Cognito ID token for the user. 'getToken()' is in 'helper.js'.
let idToken = getToken();
let loginData = {
  [COGNITO_ID]: idToken,
};

const params = {
  Bucket: "BUCKET", // The Amazon Simple Storage Solution (S3) bucket to store the
 transcriptions.
  Region: "REGION", // The AWS Region
  identityPoolID: "IDENTITY_POOL_ID", // Amazon Cognito Identity Pool ID.
};

// Create an Amazon Transcribe service client object.
const client = new TranscribeClient({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});

// Create an Amazon S3 client object.
const s3Client = new S3Client({
```

```
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});
```

Lorsque la page HTML se charge, un dossier portant le nom de l'utilisateur est `updateUserInterface` créé dans le compartiment Amazon S3 si c'est la première fois qu'il se connecte à l'application. Dans le cas contraire, il met à jour l'interface utilisateur avec les transcriptions des sessions précédentes de l'utilisateur.

```
window.onload = async () => {
  // Set the parameters.
  const userParams = {
    // Get the access token. 'GetAccessToken()' is in 'helper.js'.
    AccessToken: getAccessToken(),
  };
  // Create a CognitoIdentityProviderClient client object.
  const client = new CognitoIdentityProviderClient({ region: params.Region });
  try {
    const data = await client.send(new GetUserCommand(userParams));
    const username = data.Username;
    // Export username for use in 'recorder.js'.
    exports.username = username;
    try {
      // If this is user's first sign-in, create a folder with user's name in Amazon S3
 bucket.
      // Otherwise, no effect.
      const Key = `${username}/`;
      try {
        const data = await s3Client.send(
          new PutObjectCommand({ Key: Key, Bucket: params.Bucket })
        );
        console.log("Folder created for user ", data.Username);
      } catch (err) {
        console.log("Error", err);
      }
      try {
        // Get a list of the objects in the Amazon S3 bucket.
        const data = await s3Client.send(
```

```
          new ListObjectsCommand({ Bucket: params.Bucket, Prefix: username })
        );
        // Create a variable for the list of objects in the Amazon S3 bucket.
        const output = data.Contents;
        // Loop through the objects, populating a row on the user interface for each
object.
        for (var i = 0; i < output.length; i++) {
          var obj = output[i];
          const objectParams = {
            Bucket: params.Bucket,
            Key: obj.Key,
          };
          // Get the name of the object from the Amazon S3 bucket.
          const data = await s3Client.send(new GetObjectCommand(objectParams));
          // Extract the body contents, a readable stream, from the returned data.
          const result = data.Body;
          // Create a variable for the string version of the readable stream.
          let stringResult = "";
          // Use 'yieldUnit8Chunks' to convert the readable streams into JSON.
          for await (let chunk of yieldUint8Chunks(result)) {
            stringResult += String.fromCharCode.apply(null, chunk);
          }
          // The setTimeout function waits while readable stream is converted into
JSON.
          setTimeout(function () {
            // Parse JSON into human readable transcript, which will be displayed on
user interface (UI).
            const outputJSON =
              JSON.parse(stringResult).results.transcripts[0].transcript;
            // Create name for transcript, which will be displayed.
            const outputJSONTime = JSON.parse(stringResult)
              .jobName.split("/")[0]
              .replace("-job", "");
            i++;
            //
            // Display the details for the transcription on the UI.
            // 'displayTranscriptionDetails()' is in 'helper.js'.
            displayTranscriptionDetails(
              i,
              outputJSONTime,
              objectParams.Key,
              outputJSON
            );
          }, 1000);
```

```
        }
      } catch (err) {
        console.log("Error", err);
      }
    } catch (err) {
      console.log("Error creating presigned URL", err);
    }
  } catch (err) {
    console.log("Error", err);
  }
};

// Convert readable streams.
async function* yieldUint8Chunks(data) {
  const reader = data.getReader();
  try {
    while (true) {
      const { done, value } = await reader.read();
      if (done) return;
      yield value;
    }
  } finally {
    reader.releaseLock();
  }
}
```

Lorsque l'utilisateur enregistre un message vocal à des fins de transcription, il `upload` télécharge les enregistrements dans le compartiment Amazon S3. Cette fonction est appelée depuis le `recorder.js` fichier.

```
// Upload recordings to Amazon S3 bucket
window.upload = async function (blob, userName) {
  // Set the parameters for the recording recording.
  const Key = `${userName}/test-object-${Math.ceil(Math.random() * 10 ** 10)}`;
  let signedUrl;

  // Create a presigned URL to upload the transcription to the Amazon S3 bucket when it
 is ready.
  try {
    // Create an Amazon S3RequestPresigner object.
    const signer = new S3RequestPresigner({ ...s3Client.config });
```

```
    // Create the request.
    const request = await createRequest(
      s3Client,
      new PutObjectCommand({ Key, Bucket: params.Bucket })
    );
    // Define the duration until expiration of the presigned URL.
    const expiration = new Date(Date.now() + 60 * 60 * 1000);
    // Create and format the presigned URL.
    signedUrl = formatUrl(await signer.presign(request, expiration));
    console.log(`\nPutting "${Key}"`);
  } catch (err) {
    console.log("Error creating presigned URL", err);
  }
  try {
    // Upload the object to the Amazon S3 bucket using a presigned URL.
    response = await fetch(signedUrl, {
      method: "PUT",
      headers: {
        "content-type": "application/octet-stream",
      },
      body: blob,
    });
    // Create the transcription job name. In this case, it's the current date and time.
    const today = new Date();
    const date =
      today.getFullYear() +
      "-" +
      (today.getMonth() + 1) +
      "-" +
      today.getDate();
    const time =
      today.getHours() + "-" + today.getMinutes() + "-" + today.getSeconds();
    const jobName = date + "-time-" + time;

    // Call the "createTranscriptionJob()" function.
    createTranscriptionJob(
      "s3://" + params.Bucket + "/" + Key,
      jobName,
      params.Bucket,
      Key
    );
  } catch (err) {
    console.log("Error uploading object", err);
  }
```

```
};

// Create the AWS Transcribe transcription job.
const createTranscriptionJob = async (recording, jobName, bucket, key) => {
  // Set the parameters for transcriptions job
  const params = {
    TranscriptionJobName: jobName + "-job",
    LanguageCode: "en-US", // For example, 'en-US',
    OutputBucketName: bucket,
    OutputKey: key,
    Media: {
      MediaFileUri: recording, // For example, "https://transcribe-demo.s3-
REGION.amazonaws.com/hello_world.wav"
    },
  };
  try {
    // Start the transcription job.
    const data = await client.send(new StartTranscriptionJobCommand(params));
    console.log("Success - transcription submitted", data);
  } catch (err) {
    console.log("Error", err);
  }
};
```

`deleteTranscription`supprime une transcription de l'interface utilisateur et `deleteRow` supprime une transcription existante du compartiment Amazon S3. Les deux sont déclenchés par le bouton Supprimer de l'interface utilisateur.

```
// Delete a transcription from the Amazon S3 bucket.
window.deleteJSON = async (jsonFileName) => {
  try {
    await s3Client.send(
      new DeleteObjectCommand({
        Bucket: params.Bucket,
        Key: jsonFileName,
      })
    );
    console.log("Success - JSON deleted");
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
// Delete a row from the user interface.
window.deleteRow = function (rowid) {
  const row = document.getElementById(rowid);
  row.parentNode.removeChild(row);
};
```

Enfin, exécutez la commande suivante à l'invite de commande JavaScript pour regrouper le fichier correspondant à cet exemple dans un fichier nommé `main.js` :

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

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

Exécutez l'application

Vous pouvez ensuite consulter l'application à l'emplacement ci-dessous.

```
DOMAIN/login?
client_id=APP_CLIENT_ID&response_type=token&scope=aws.cognito.signin.user.admin+email
+openid+phone+profile&redirect_uri=REDIRECT_URL
```

Amazon Cognito facilite l'exécution de l'application en fournissant un lien dans la console de gestion des services AWS Web. Accédez simplement aux paramètres du client d'application de votre groupe d'utilisateurs Amazon Cognito et sélectionnez l'interface utilisateur Launch Hosted. L'URL de l'application est au format suivant.

> **Important**
>
> L'interface utilisateur hébergée utilise par défaut un type de réponse « code ». Cependant, ce didacticiel est conçu pour le type de réponse « jeton », vous devez donc le modifier.

Supprimer les AWS ressources

Lorsque vous avez terminé le didacticiel, vous devez supprimer les ressources afin de ne pas encourir de frais inutiles. Comme vous avez ajouté du contenu aux deux compartiments Amazon S3, vous devez les supprimer manuellement. Vous pouvez ensuite supprimer les ressources restantes à l'aide de la console de gestion des services AWS Web ou du AWS CLI. Les instructions sur la façon de modifier la pile ou de supprimer la pile et ses ressources associées une fois le didacticiel terminé, voir ici GitHub.

Invoquer Lambda avec API Gateway

Vous pouvez appeler une fonction Lambda à l'aide d'Amazon API Gateway, un AWS service permettant de créer, de publier, de gérer, de surveiller et de sécuriser REST, HTTP et des WebSocket API à grande échelle. Les développeurs d'API peuvent créer des API qui accèdent à AWS ou à d'autres services web, ainsi qu'aux données stockées dans le cloud AWS. En tant que développeur d'API Gateway, vous pouvez créer des API à utiliser dans vos propres applications clientes. Pour plus d'informations, consultez Qu'est-ce qu'Amazon API Gateway ?

AWS Lambdaest 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 sur AWS Lambda, consultez Qu'est-ce qu'un AWS Lambda.

Dans cet exemple, vous créez une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple invoque différents AWS services permettant d'effectuer 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.

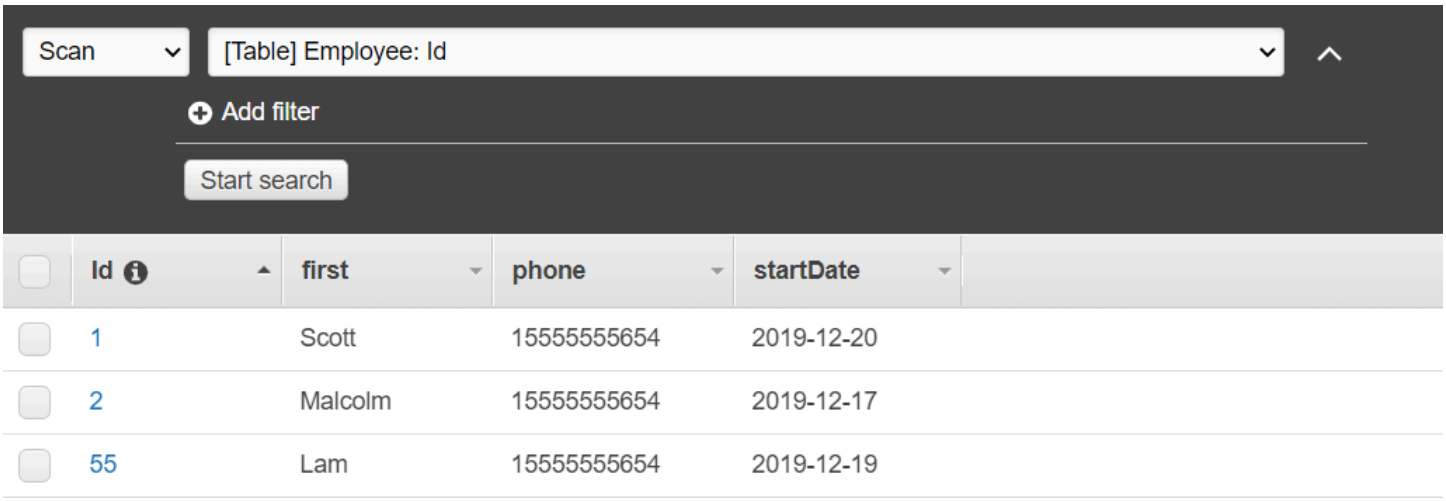
Cet exemple montre comment utiliser la JavaScript logique pour créer une solution qui exécute ce cas d'utilisation. Par exemple, vous apprendrez à lire une base de données pour déterminer quels employés ont atteint le premier anniversaire, à traiter les données et à envoyer un message texte

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 : 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 :

1. Prérequis complets

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 for 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 des SDK et des outils.

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, voir 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 AWS CloudFormation comme décrit dans ce didacticiel.

Créez les AWS ressources à l'aide de AWS CloudFormation

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

Pour créer la AWS CloudFormation pile à l'aide de AWS CLI :

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

   > **Note**
   >
   > Le AWS CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour en savoir plus sur le AWS CDK, veuillez consulter le Guide du développeur AWS Cloud Development Kit (AWS CDK).

3. Exécutez la commande suivante depuis la ligne de commande, en remplaçant *`STACK_NAME 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 AWS 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 fonction AWS Lambda

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 AWS Region. 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 AWS Region. Ce code est disponible ici GitHub.

Importez d'abord les modules et commandes requis AWS SDK for JavaScript (v3). Calculez ensuite la date du jour et attribuez-la à un paramètre. Troisièmement, créez les paramètres pour`ScanCommand`. Remplacez *TABLE_NAME* 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.)

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

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

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
```

```
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "Employees",
};
```

Analyse de la table DynamoDB

Créez d'abord une fonction async/await 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));
    data.Items.forEach(function (element) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
            "Hi " +
            element.firstName.S +
            "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
```

```
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Regroupement de la fonction Lambda

Cette rubrique décrit comment regrouper `mylambdafunction.ts` les AWS SDK for 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 nommée`index.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.

Déployer la fonction Lambda

À la racine de votre projet, créez un `lambda-function-setup.ts` fichier et collez-y le contenu ci-dessous.

Remplacez *BUCKET_NAME* 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 de la version ZIP de votre fonction Lambda. Remplacez *ROLE* 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

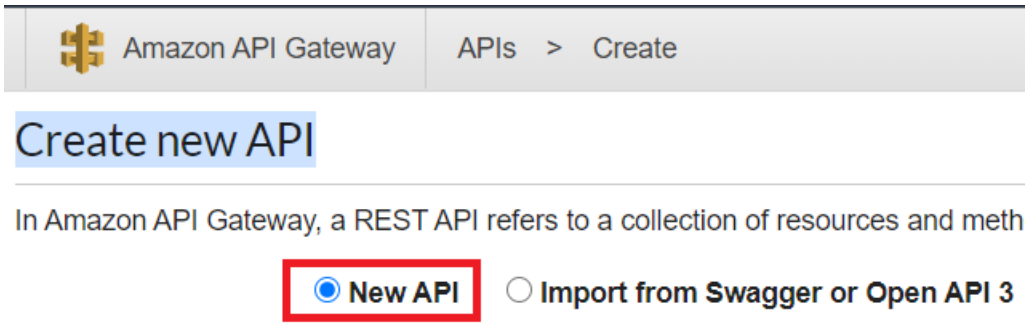
Pour créer l'application :

1. Création de l'API Rest
2. Testez la méthode API Gateway
3. Déployer la méthode API Gateway

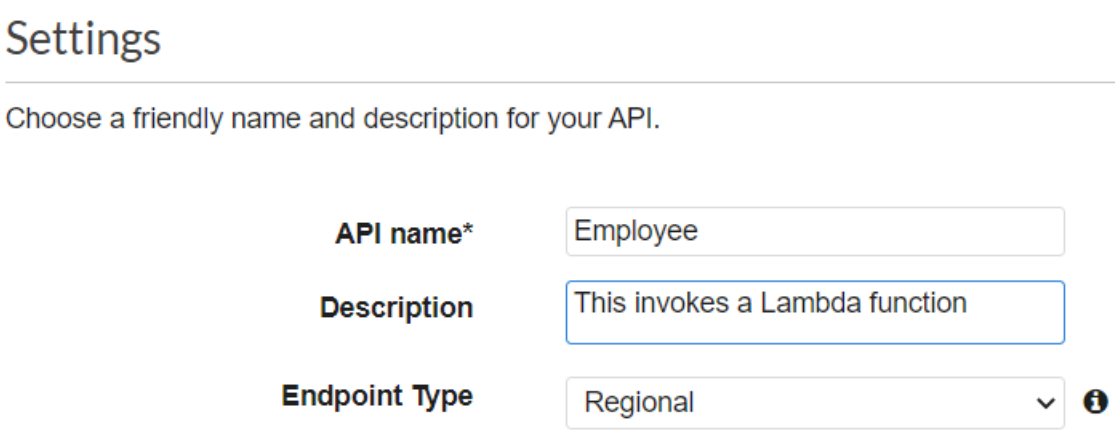
Création de l'API Rest

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.

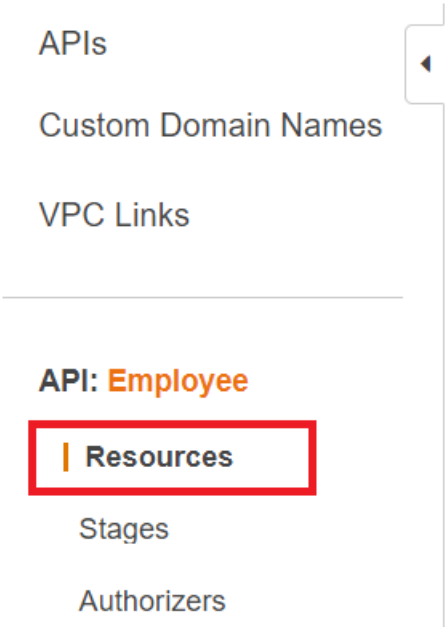
1. Connectez-vous à la console Amazon API Gateway.
2. Sous Rest API, choisissez Build.
3. Sélectionnez Nouvelle API.



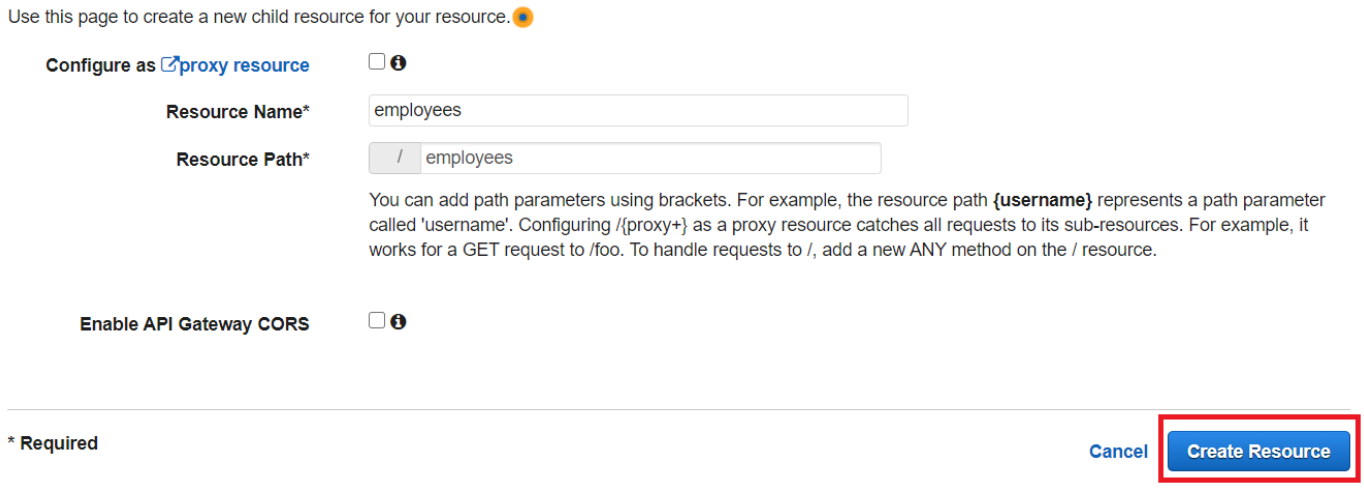
4. Spécifiez Employee comme nom d'API et fournissez une description.



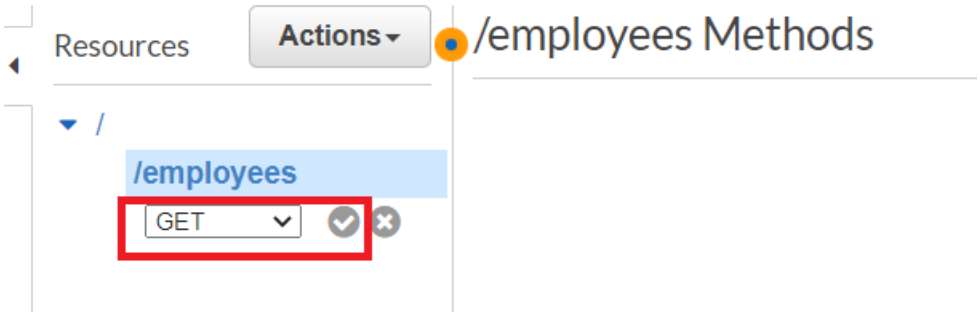
5. Sélectionnez Create API (Créer une API).
6. Choisissez Ressources dans la section Employé.



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.



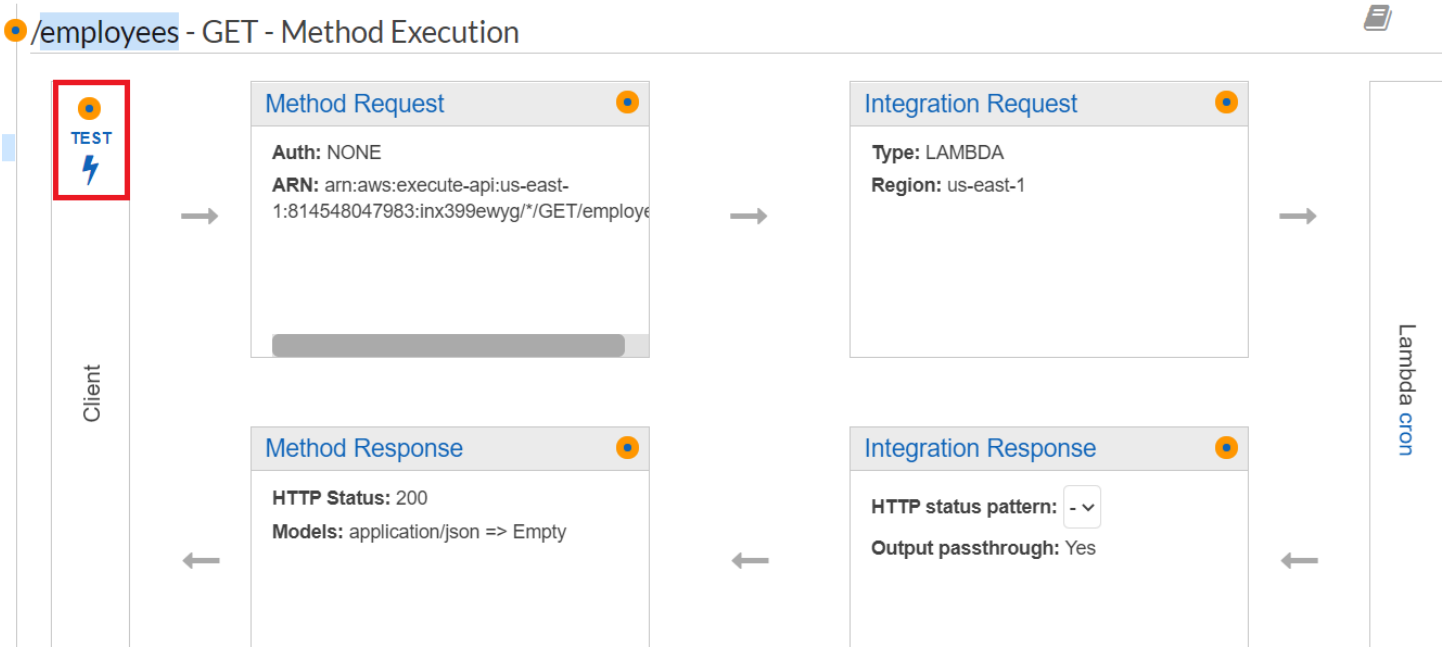
10. Choisissez /employees, sélectionnez Create Method dans les Actions, puis sélectionnez GET dans le menu déroulant situé sous /employees. Choisissez l'icône représentant une coche.



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.

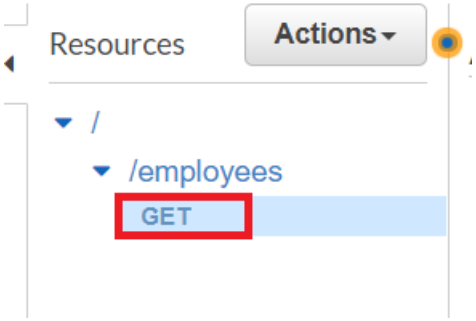


Une fois que la fonction Lambda est invoquée, vous pouvez consulter le fichier journal pour voir un message de réussite.

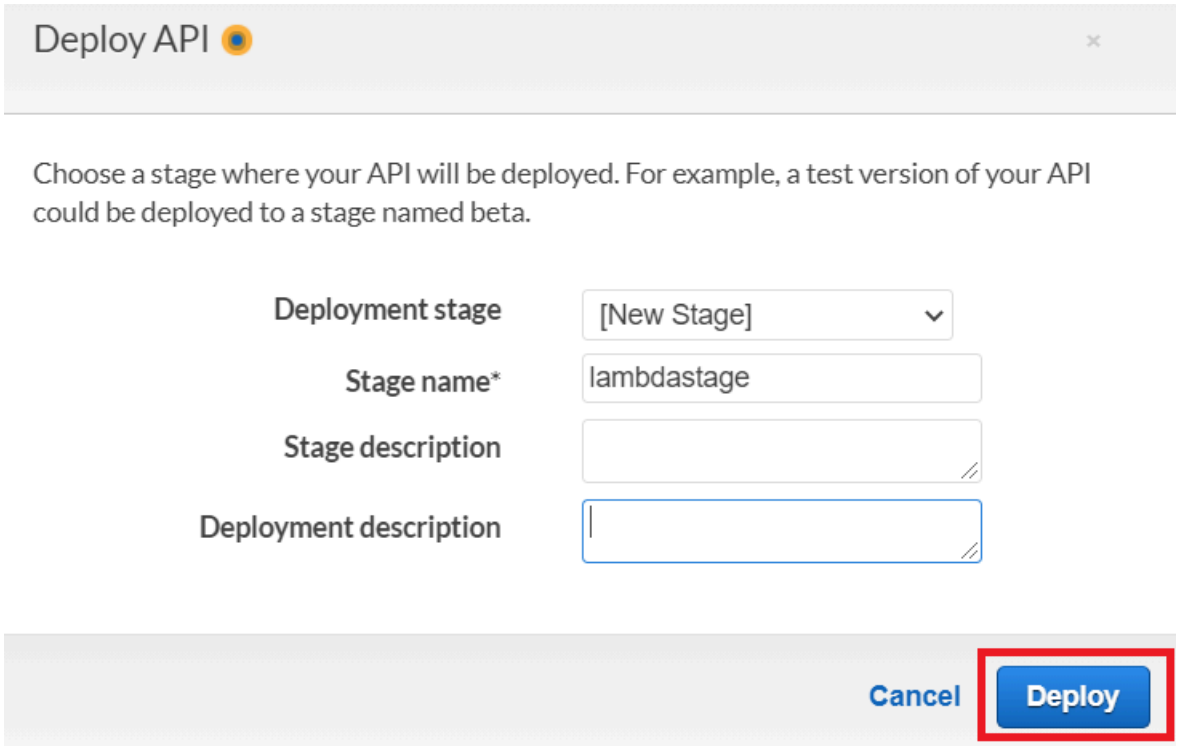
Déployer la méthode API Gateway

Une fois le test réussi, vous pouvez déployer la méthode depuis la console Amazon API Gateway.

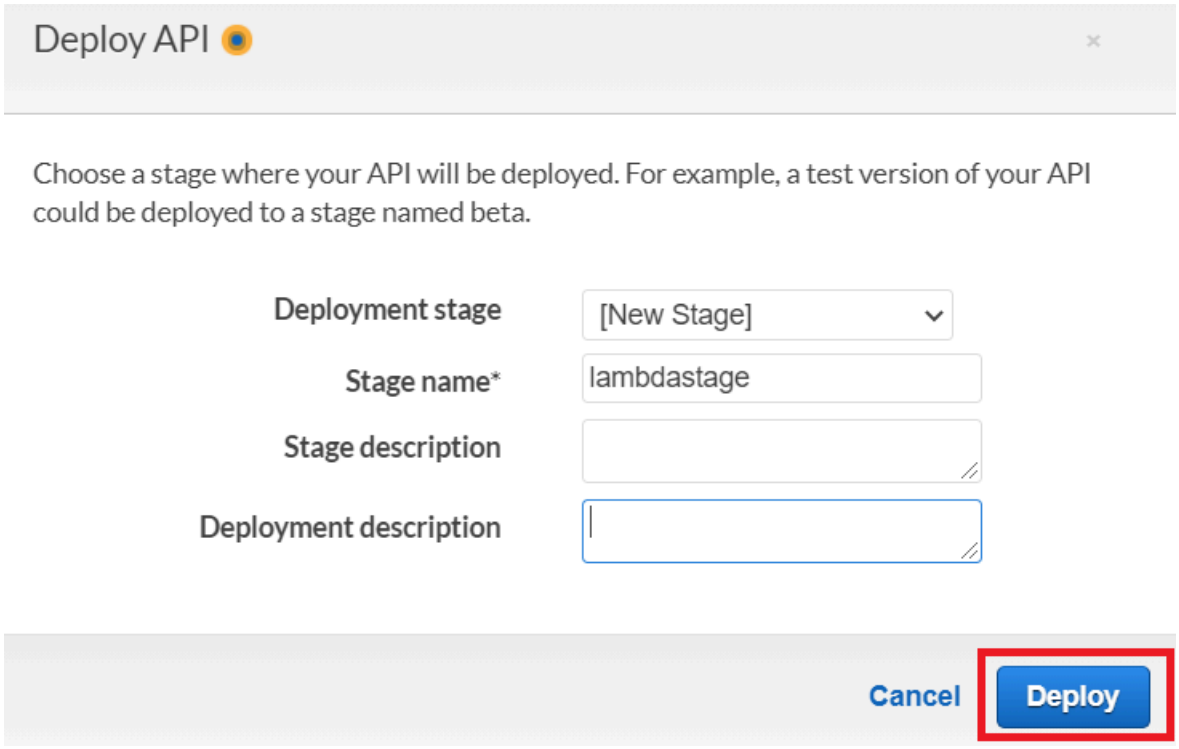
1. Choisissez Get.



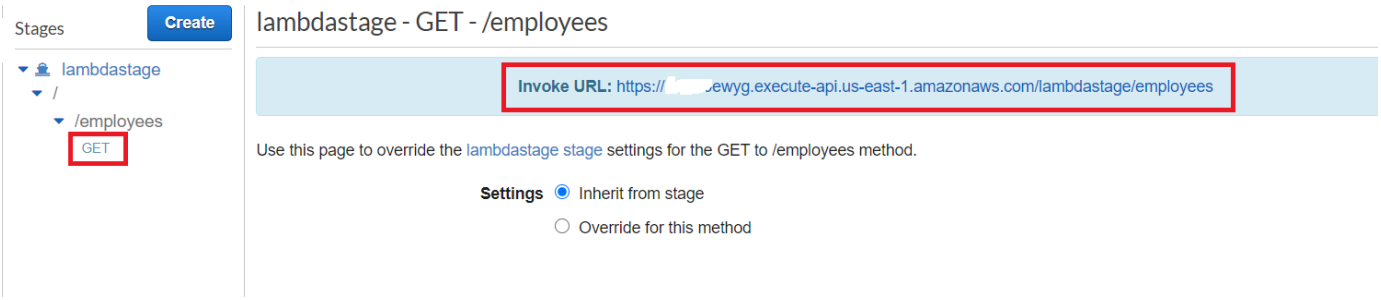
2. Dans le menu déroulant Actions, sélectionnez Déployer l'API.



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 for 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 AWS CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

1. Ouvrez le AWS CloudFormationdans la console AWS de gestion.
2. Ouvrez la page Stacks et sélectionnez la pile.
3. Sélectionnez Supprimer.

Création de flux de travail AWS sans serveur à l'aide de AWS SDK for JavaScript

Vous pouvez créer un flux de travail AWS sans serveur en utilisant Step Functions, le AWS SDK for Java et. AWS Step Functions Chaque étape du flux est mise en place à l'aide d'une fonction AWS Lambda. Lambda est un service de calcul qui vous permet d'exécuter du code sans devoir approvisionner ou gérer des serveurs. Step Functions est un service d'orchestration sans serveur qui vous permet de combiner des fonctions Lambda et d'autres services AWS afin de créer des applications métier essentielles.

> **Note**
>
> Vous pouvez créer des fonctions Lambda dans différents langages de programmation. Dans ce didacticiel, les fonctions Lambda ont été implémentées à l'aide de l'API Lambda Java. Pour plus d'informations sur Lambda, consultez Qu'est-ce que Lambda ?

Dans ce didacticiel, vous allez créer un flux de travail qui crée des tickets d'assistance pour une organisation. Chaque étape du flux de travail exécute une opération sur le ticket. Ce didacticiel explique comment JavaScript traiter les données du flux de travail. Par exemple, vous apprendrez à lire les données transmises au flux de travail, à transmettre des données entre les étapes et à invoquer des AWS services depuis le flux de travail.

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 plus débité.

Rubriques

- Tâches préalables
- Créez les AWS ressources
- Création du flux de travail
- Création des fonctions Lambda
- Ajouter les fonctions Lambda aux flux de travail
- Exécutez votre flux de travail à l'aide de la console Step Functions
- Supprimer les AWS 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 for 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 des SDK et des outils.

Créez les AWS ressources

Ce didacticiel nécessite les ressources suivantes.

- Une table Amazon DynamoDB nommée Case avec une clé nommée Id.
- Rôle IAM nommé `lambda-support` utilisé pour appeler les fonctions Lambda. Ce rôle possède des politiques qui lui permettent d'appeler les services Amazon DynamoDB et Amazon Simple Email Service à partir d'une fonction Lambda.
- Rôle IAM nommé `workflow-support` utilisé pour appeler le flux de travail.
- Un compartiment Amazon S3 pour héberger les fonctions Lambda.

Vous pouvez créer ces ressources manuellement, mais nous vous recommandons de les approvisionner à l'aide du AWS Cloud Development Kit (AWS CDK) (AWS CDK) comme décrit dans ce didacticiel.

Créez les AWS ressources à l'aide du AWS CloudFormation

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

Pour créer la AWS CloudFormation pile :

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

   > **Note**
   >
   > Le AWS CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour en savoir plus sur le AWS CDK, veuillez consulter le Guide du développeur AWS Cloud Development Kit (AWS CDK).

3. Exécutez la commande suivante depuis la ligne de commande, en remplaçant *`STACK_NAME 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 AWS CloudFormation l'utilisateur.

Créez les AWS ressources à l'aide de la console de gestion Amazon Web Services ;

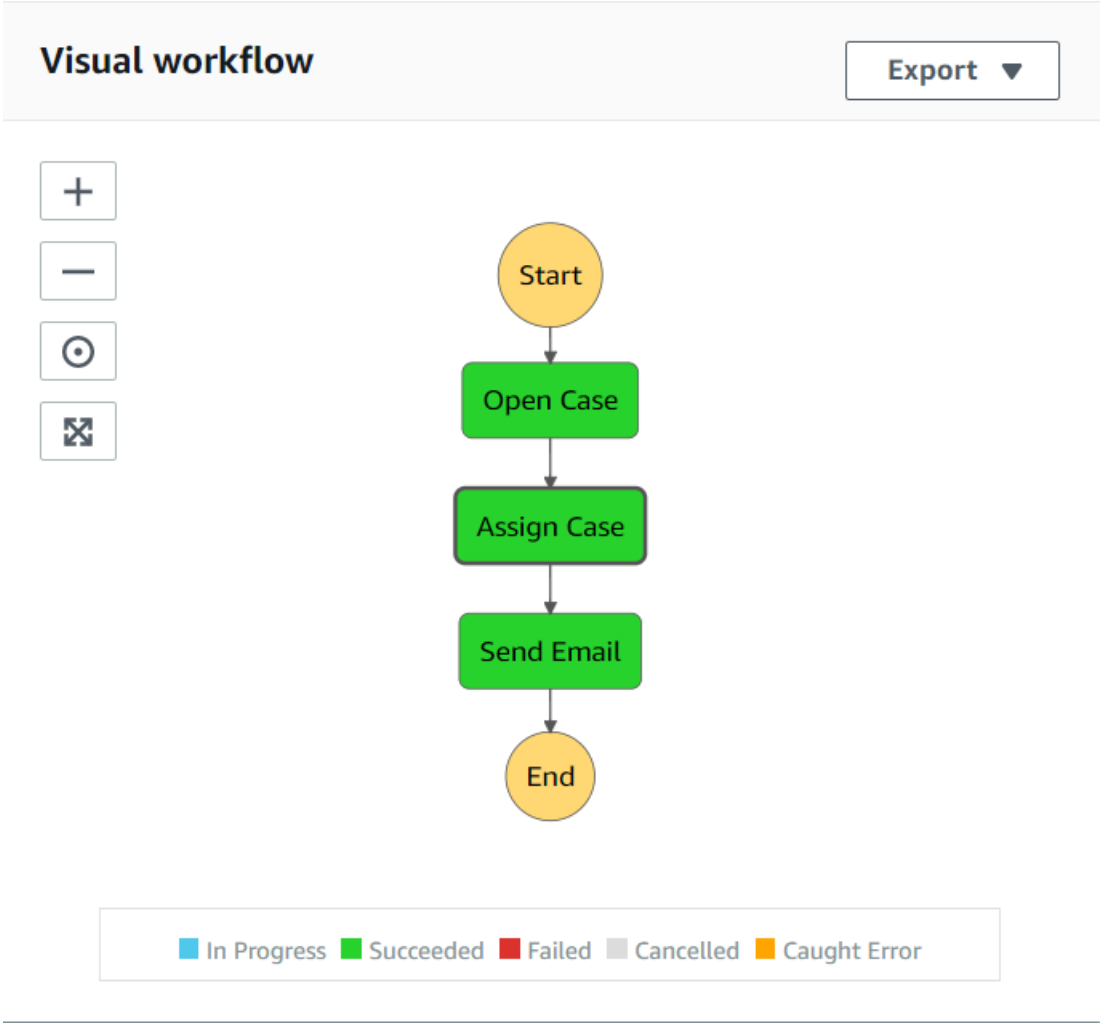
Pour créer des ressources pour l'application dans la console, suivez les instructions du guide de l'AWS CloudFormationutilisateur. Utilisez le modèle fourni pour créer un fichier nommé `setup.yaml` et copiez le contenu ici GitHub.

> **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.

Consultez la liste des ressources dans la console en ouvrant la pile sur le AWS CloudFormation tableau de bord et en choisissant l'onglet Ressources. Vous en avez besoin pour le didacticiel.

Création du flux de travail

La figure suivante montre le flux de travail que vous allez créer avec ce didacticiel.



Voici ce qui se passe à chaque étape du flux de travail :

+ Démarrer - Lance le flux de travail.

+ Open Case — Gère la valeur d'un identifiant de ticket d'assistance en la transmettant au flux de travail.

+ Affecter un dossier : attribue le dossier d'assistance à un employé et stocke les données dans une table DynamoDB.

+ Envoyer un e-mail — Envoie un e-mail à l'employé en utilisant Amazon Simple Email Service (Amazon SES) pour l'informer de l'existence d'un nouveau ticket.

+ Fin : arrête le flux de travail.

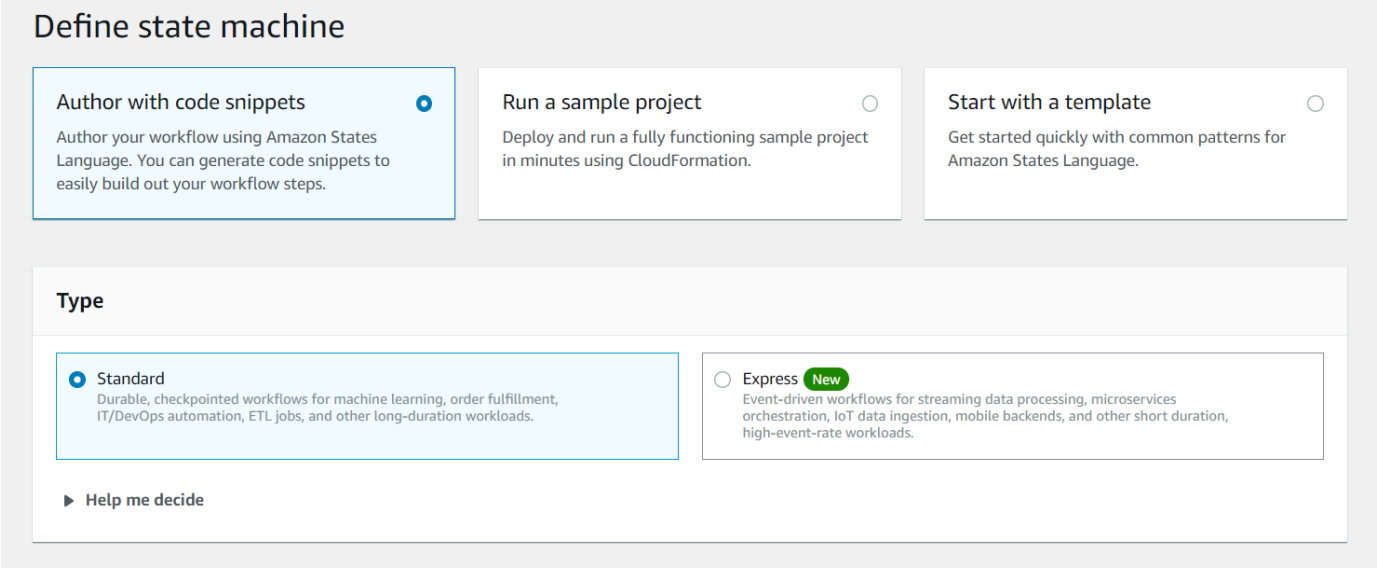
Créez un flux de travail sans serveur à l'aide des fonctions Step

Vous pouvez créer un flux de travail qui traite les tickets d'assistance. Pour définir un flux de travail à l'aide de Step Functions, vous devez créer un document Amazon States Language (basé sur JSON) pour définir votre machine à états. Un document Amazon States Language décrit chaque étape. Après avoir défini le document, Step Functions fournit une représentation visuelle du flux de travail. La figure suivante montre le document Amazon States Language et la représentation visuelle du flux de travail.

Les flux de travail peuvent transmettre des données entre les étapes. Par exemple, l'étape Ouvrir un dossier traite une valeur d'identifiant de dossier (transmise au flux de travail) et transmet cette valeur à l'étape Attribuer un dossier. Plus loin dans ce didacticiel, vous allez créer une logique d'application dans la fonction Lambda pour lire et traiter les valeurs des données.

Pour créer un flux de travail

1. Ouvrez la console Amazon Web Services.
2. Choisissez Créer une machine d'état.
3. Choisissez Auteur avec des extraits de code. Dans la zone Type, sélectionnez Standard.



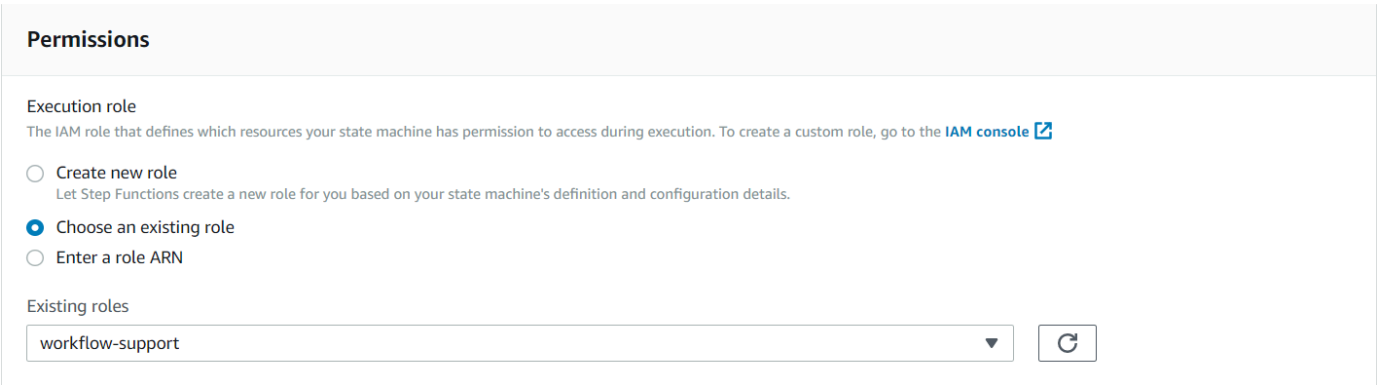
4. Spécifiez le document Amazon States Language en saisissant le code suivant.

```
{
"Comment": "A simple AWS Step Functions state machine that automates a call center
 support session.",
"StartAt": "Open Case",
"States": {
```

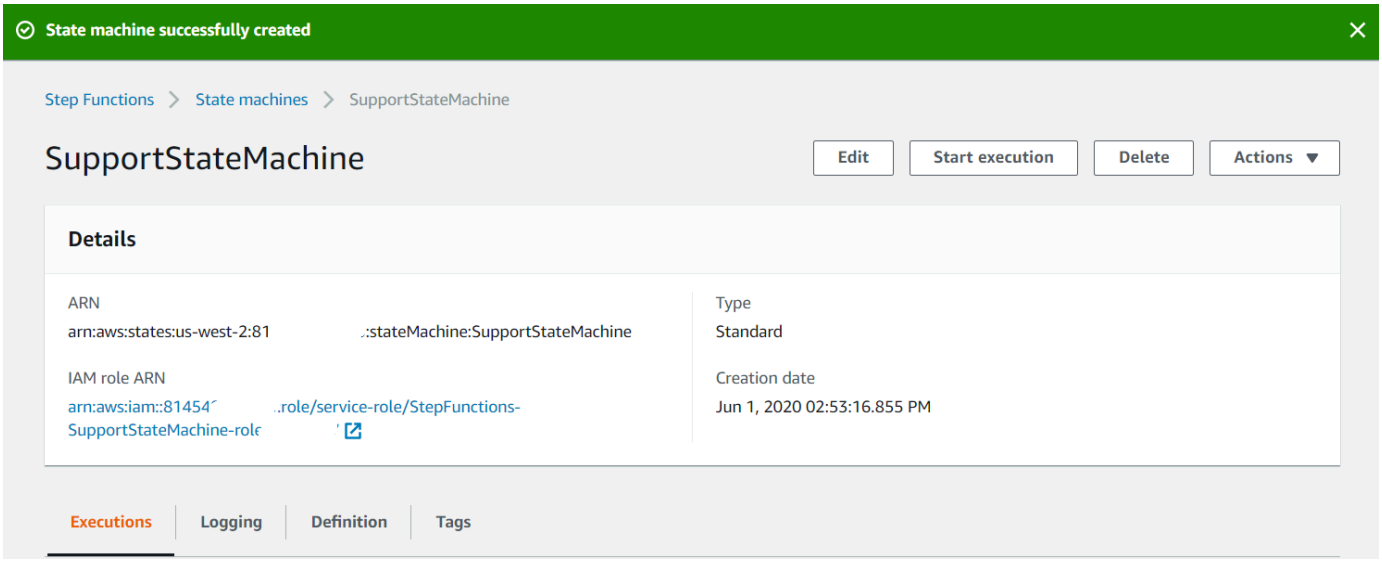
```
"Open Case": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"Next": "Assign Case"
},
"Assign Case": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"Next": "Send Email"
},
"Send Email": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"End": true
}
}
}
```

> **Note**
>
> Ne vous inquiétez pas des erreurs liées aux valeurs des ressources Lambda. Vous mettrez à jour ces valeurs ultérieurement dans ce didacticiel.

5. Choisissez Suivant.
6. Dans le champ du nom, entrez SupportStateMachine.
7. Dans la section Autorisation, choisissez Choisir un rôle existant.
8. Choisissez workflow-support (le rôle IAM que vous avez créé).



9. Choisissez Create state machine (Créer une machine d'état). Un message s'affiche indiquant que la machine à états a été créée avec succès.



Création des fonctions Lambda

Utilisez l'API d'exécution Lambda pour créer les fonctions Lambda. Dans cet exemple, trois étapes de flux de travail correspondent chacune à chaque fonction Lambda.

Créez ces fonctions Lambda, comme décrit dans les sections suivantes :

- Fonction Lambda GetID- Utilisé comme première étape du flux de travail qui traite la valeur de l'identifiant du ticket.
- Classe Lambda AddItem- Utilisé comme deuxième étape du flux de travail qui attribue le ticket à un employé et stocke les données dans une base de données DynamoDB.
- classe Lambda sendemail- Utilisé comme troisième étape du flux de travail qui utilise Amazon SES pour envoyer un e-mail à l'employé afin de l'informer de l'existence du ticket.

Fonction Lambda GetID

Créez une fonction Lambda qui renvoie la valeur de l'ID de ticket transmise à la deuxième étape du flux de travail.

```
exports.handler = async (event) => {
// Create a support case using the input as the case ID, then return a confirmation
 message
try{
    const myCaseID = event.inputCaseID;
```

```
    var myMessage = "Case " + myCaseID + ": opened...";
    var result = { Case: myCaseID, Message: myMessage };
   }
catch(err){
    console.log('Error', err);
   }
};
```

Entrez ce qui suit dans la ligne de commande pour utiliser webpack afin de regrouper le fichier dans un fichier nomméindex.js.

```
webpack getid.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

Compressez ensuite index.js dans un nom de ZIP fichiergetid.js.zip. Téléchargez le ZIP fichier dans le compartiment Amazon S3 que vous avez créé dans la rubrique de cet exemple.

Cet exemple de code est disponible ici GitHub.

Classe Lambda AddItem

Créez une fonction Lambda qui sélectionne un employé à qui attribuer le ticket, puis stocke les données du ticket dans une table DynamoDB nommée Case.

```
"use strict";
// Load the required clients and commands.
const { PutItemCommand } = require ( "@aws-sdk/client-dynamodb" );
const { dynamoClient } = require ( "../libs/dynamoClient" );

exports.handler = async (event) => {
  try {
    // Helper function to send message using Amazon SNS.
    const val = event;
    //PersistCase adds an item to a DynamoDB table
    const tmp = Math.random() <= 0.5 ? 1 : 2;
    console.log(tmp);
    if (tmp == 1) {
      const params = {
        TableName: "Case",
        Item: {
          id: { N: val.Case },
```

```
          empEmail: { S: "brmur@amazon.com" },
          name: { S: "Tom Blue" },
        },
      };
      console.log("adding item for tom");
      try {
        const data = await dynamoClient.send(new PutItemCommand(params));
        console.log(data);
      } catch (err) {
        console.error(err);
      }
      var result = { Email: params.Item.empEmail };
      return result;
    } else {
      const params = {
        TableName: "Case",
        Item: {
          id: { N: val.Case },
          empEmail: { S: "RECEIVER_EMAIL_ADDRESS" }, // Valid Amazon Simple
 Notification Services (Amazon SNS) email address.
          name: { S: "Sarah White" },
        },
      };
      console.log("adding item for sarah");
      try {
        const data = await dynamoClient.send(new PutItemCommand(params));
        console.log(data);
      } catch (err) {
        console.error(err);
      }
      return params.Item.empEmail;
      var result = { Email: params.Item.empEmail };
    }
  } catch (err) {
    console.log("Error", err);
  }
};
```

Entrez ce qui suit dans la ligne de commande pour utiliser webpack afin de regrouper le fichier dans un fichier nomméindex.js.

```
webpack additem.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

Compressez ensuite `index.js` dans un nom de `ZIP` fichier`additem.js.zip`. Téléchargez le `ZIP` fichier dans le compartiment Amazon S3 que vous avez créé dans la rubrique de cet exemple.

Cet exemple de code est disponible ici GitHub.

classe Lambda sendemail

Créez une fonction Lambda qui envoie un e-mail pour les informer du nouveau ticket. L'adresse e-mail transmise lors de la deuxième étape est utilisée.

```
// Load the required clients and commands.
const { SendEmailCommand } = require ( "@aws-sdk/client-ses" );
const { sesClient } = require ( "../libs/sesClient" );

exports.handler = async (event) => {
  // Enter a sender email address. This address must be verified.
  const senderEmail = "SENDER_EMAIL"
  const sender = "Sender Name <" + senderEmail + ">";

  // AWS Step Functions passes the  employee's email to the event.
  // This address must be verified.
  const recepient = event.S;

  // The subject line for the email.
  const subject = "New case";

  // The email body for recipients with non-HTML email clients.
  const body_text =
    "Hello,\r\n" + "Please check the database for new ticket assigned to you.";

  // The HTML body of the email.
  const body_html = `<html><head></head><body><h1>Hello!</h1><p>Please check the
 database for new ticket assigned to you.</p></body></html>`;

  // The character encoding for the email.
  const charset = "UTF-8";
  var params = {
    Source: sender,
    Destination: {
      ToAddresses: [recepient],
    },
    Message: {
      Subject: {
```

```
        Data: subject,
        Charset: charset,
      },
      Body: {
        Text: {
          Data: body_text,
          Charset: charset,
        },
        Html: {
          Data: body_html,
          Charset: charset,
        },
      },
    },
  };
  try {
    const data = await sesClient.send(new SendEmailCommand(params));
    console.log(data);
  } catch (err) {
    console.error(err);
  }
};
```

Entrez ce qui suit dans la ligne de commande pour utiliser webpack afin de regrouper le fichier dans un fichier nomméindex.js.

```
webpack sendemail.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

Compressez ensuite index.js dans un nom de ZIP fichiersendemail.js.zip. Téléchargez le ZIP fichier dans le compartiment Amazon S3 que vous avez créé dans la rubrique de cet exemple.

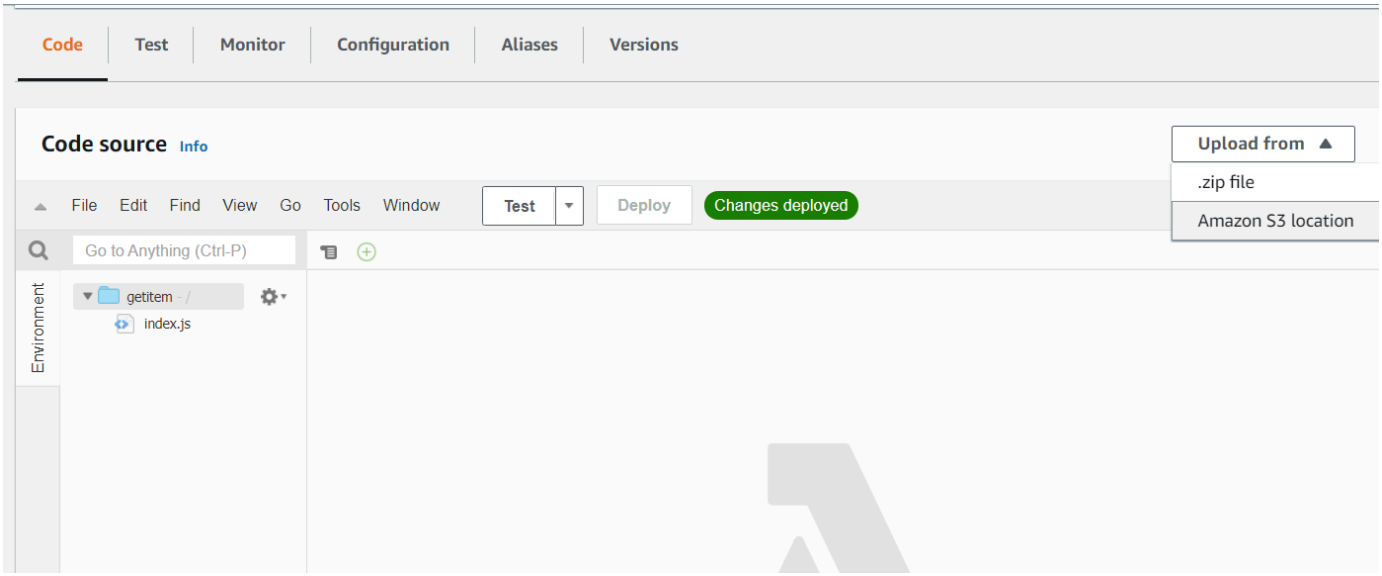
Cet exemple de code est disponible ici GitHub.

Déployer les fonctions Lambda

Pour déployer la fonction getid Lambda, procédez comme suit :

1. Ouvrez la console Lambda sur la console Amazon Web Services.
2. Choisissez Create Function (Créer une fonction).
3. Choisissez Créer à partir de zéro.

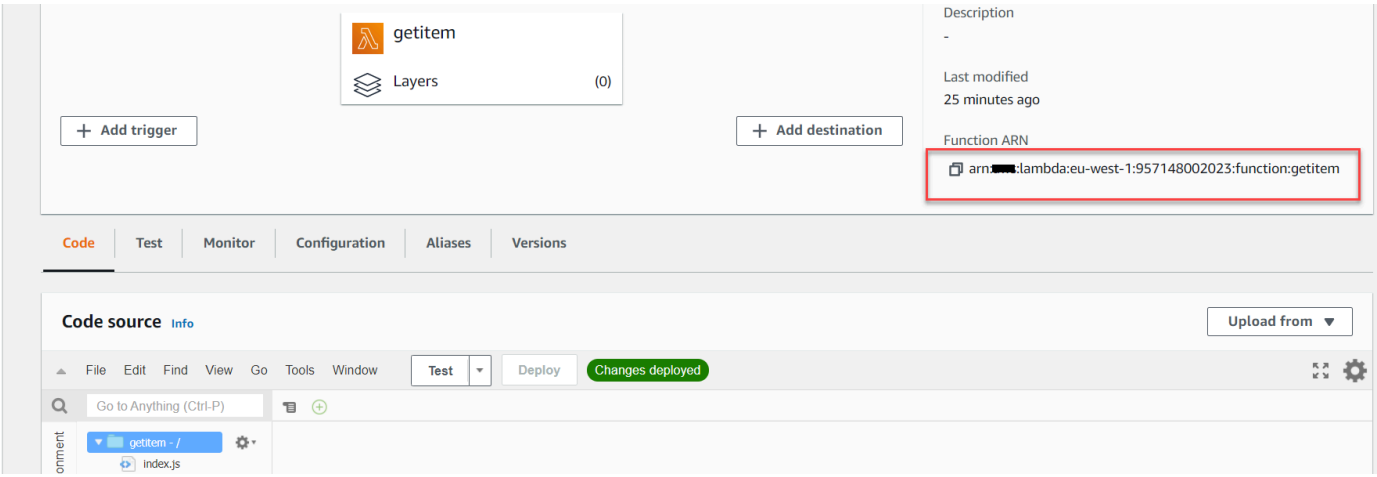
4. Dans la section Informations de base, entrez getid comme nom.
5. Dans le Runtime, choisissez Node.js 14x.
6. Choisissez Utiliser un rôle existant, puis choisissez lambda-support (le rôle IAM que vous avez créé dans le).
7. Sélectionnez Create function (Créer une fonction).
8. choisissez Upload from - Amazon S3 location.
9. Choisissez Upload, choisissez Upload from - Amazon S3 location, puis entrez l'URL du lien Amazon S3.



10. Choisissez Enregistrer.
11. Répétez cette procédure pour les nouvelles fonctions Lambda additem.js.zip et sendemail.js.zip. Lorsque vous aurez terminé, vous disposerez de trois fonctions Lambda auxquelles vous pourrez faire référence dans le document Amazon States Language.

Ajouter les fonctions Lambda aux flux de travail

1. Ouvrez la console Lambda. Notez que vous pouvez consulter la valeur Lambda Amazon Resource Name (ARN) dans le coin supérieur droit.



2. Copiez la valeur, puis collez-la à l'étape 1 du document Amazon States Language, situé dans la console Step Functions.
3. Mettez à jour la ressource pour les étapes Attribuer un dossier et Envoyer un e-mail. C'est ainsi que vous pouvez intégrer des fonctions Lambda créées à l'aide du AWS SDK pour Java dans un flux de travail créé à l'aide de Step Functions.

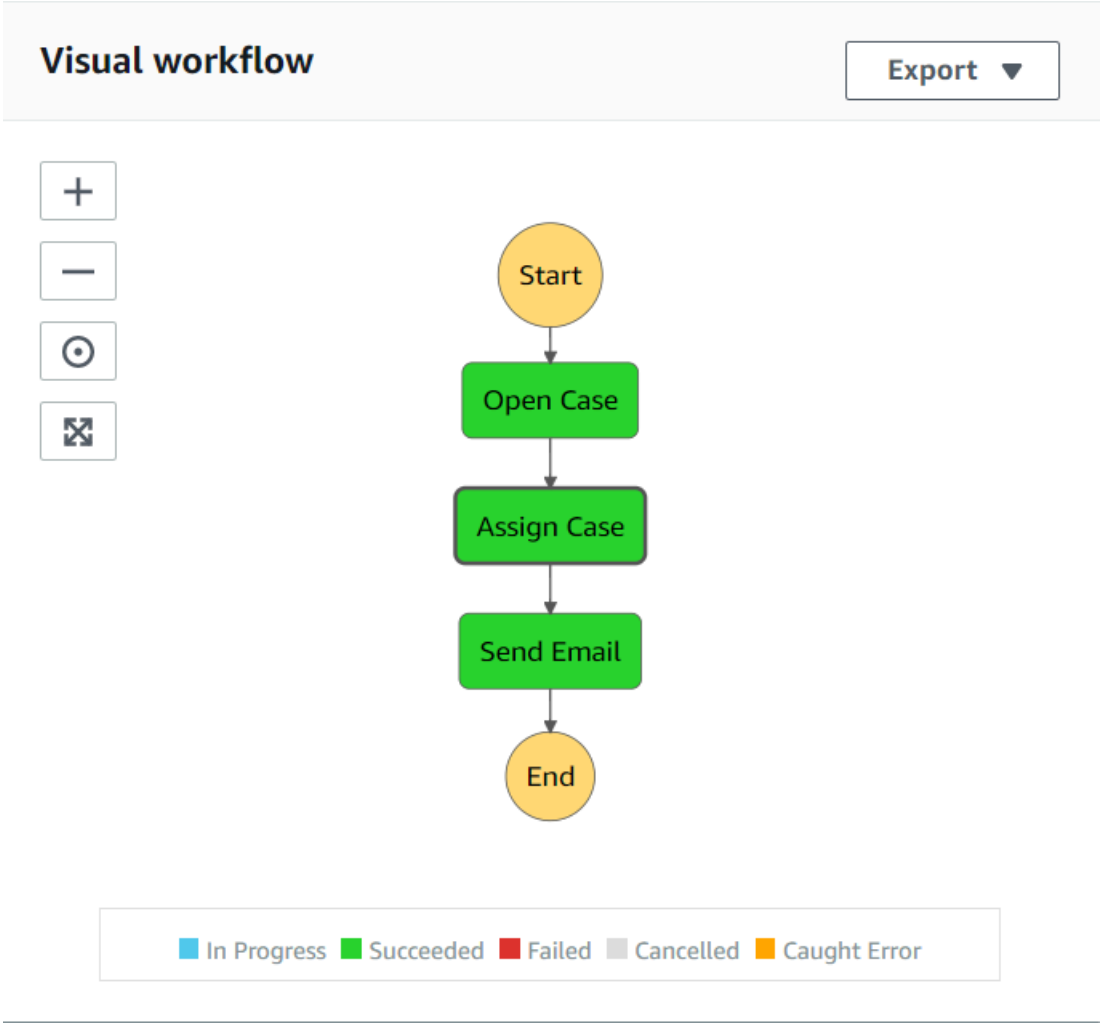
Exécutez votre flux de travail à l'aide de la console Step Functions

Vous pouvez appeler le flux de travail sur la console Step Functions. Une exécution reçoit une entrée JSON. Dans cet exemple, vous pouvez transmettre les données JSON suivantes au flux de travail.

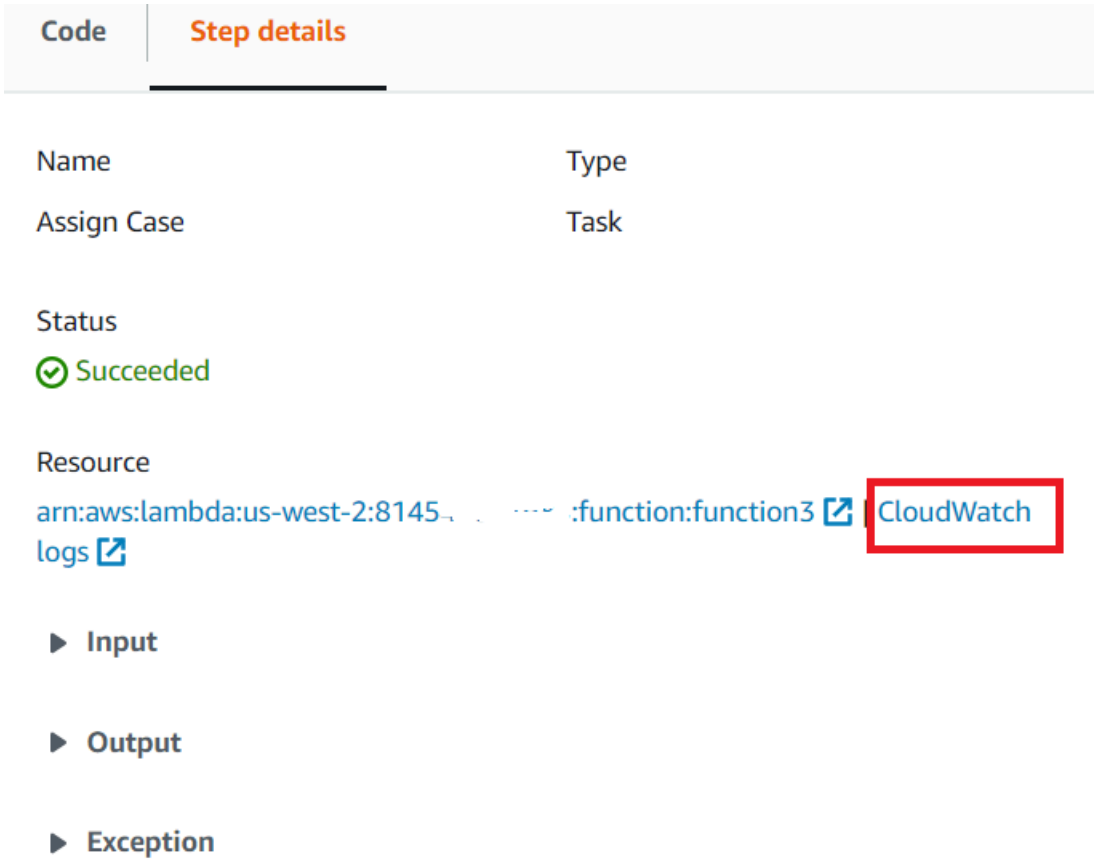
```
{
"inputCaseID": "001"
}
```

Pour exécuter votre flux de travail :

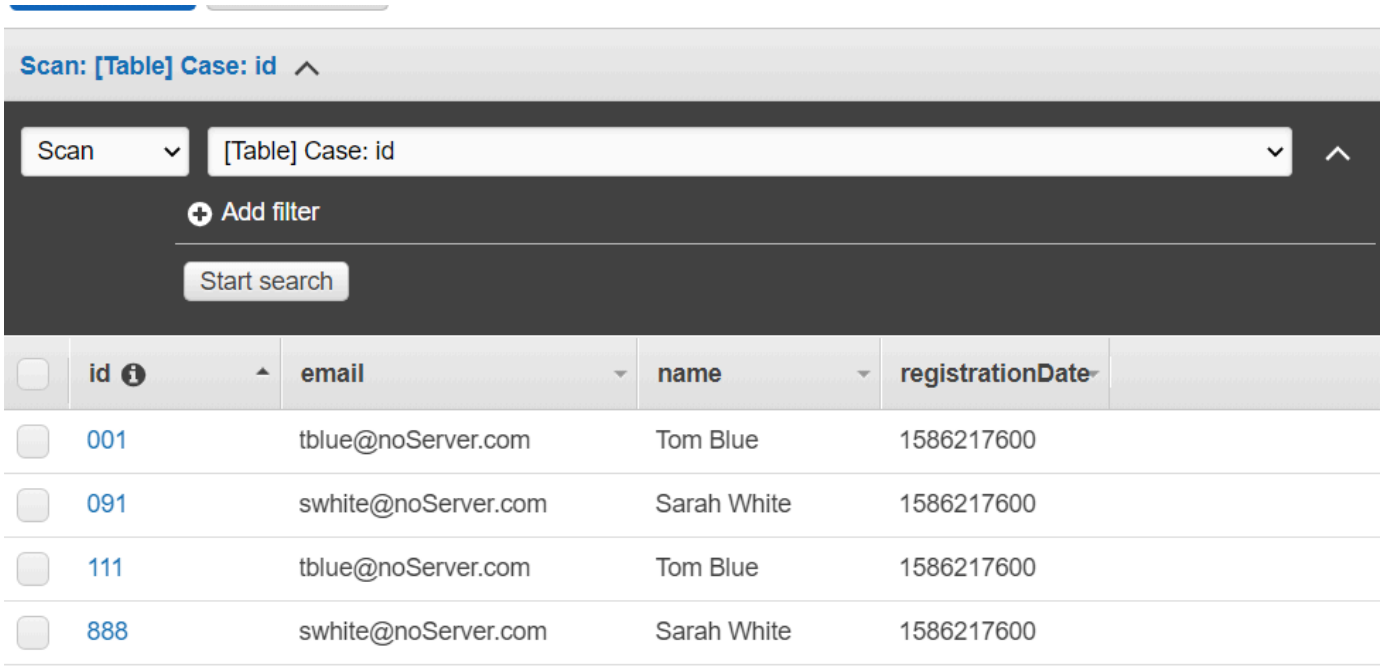
1. Sur la console Step Functions, choisissez Start execution.
2. Dans la section Input, transmettez les données JSON. Consultez le flux de travail. Au fur et à mesure que chaque étape est terminée, elle devient verte.



3. Si l'étape devient rouge, une erreur s'est produite. Vous pouvez cliquer sur l'étape et consulter les journaux accessibles depuis le côté droit.



Lorsque le flux de travail est terminé, vous pouvez afficher les données dans la table DynamoDB.



Supprimer les AWS ressources

Félicitations, vous avez créé un flux de travail AWS sans serveur à l'aide du AWS SDK for Java. 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 AWS CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

1. Ouvrez le AWS CloudFormationdans 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 Lambdaest 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 sur AWS Lambda, consultez Qu'est-ce qu'un AWS Lambda.

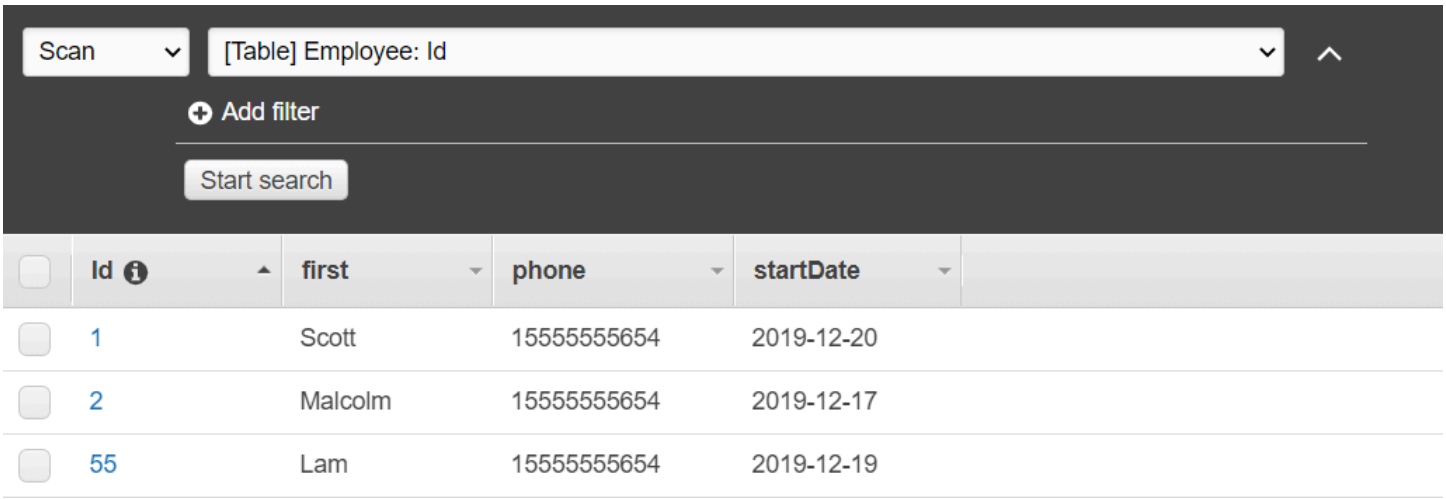
Dans ce didacticiel, vous allez créer une fonction Lambda à l'aide de l'API d'exécution JavaScript Lambda. Cet exemple invoque différents AWS services permettant d'effectuer un cas d'utilisation spécifique. Supposons, par exemple, qu'une organisation envoie un message texte mobile à ses employés pour 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 : 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 :

1. Prérequis complets
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 for 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 des SDK et des outils.

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, voir 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 AWS CloudFormation décrite dans ce didacticiel.

Créez les AWS ressources en utilisant AWS CloudFormation

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

Pour créer la AWS CloudFormation pile à l'aide de AWS CLI :

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

   > **Note**
   >
   > Le AWS CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour en savoir plus sur le AWS CDK, veuillez consulter le Guide du développeur AWS Cloud Development Kit (AWS CDK).

3. Exécutez la commande suivante depuis la ligne de commande, en remplaçant *`STACK_NAME 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 AWS CloudFormation l'utilisateur.

   Consultez la liste des ressources dans la console en ouvrant la pile sur le AWS 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 for 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 fonction AWS Lambda

Configuration du kit SDK

Importez d'abord les modules et commandes AWS SDK for JavaScript (v3) requis : `ScanCommand` DynamoDB `DynamoDBClient` et la commande Amazon `SNSClient` SNS. `PublishCommand` Remplacez *`REGION`* par AWS Region. Calculez ensuite la date du jour et attribuez-la à un paramètre. Créez ensuite les paramètres pour le `ScanCommand` fichier .Replace *`TABLE_NAME`* 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.)

```
"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 fonction async/await 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 for 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 index.js groupé 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.

Remplacez *`BUCKET_NAME`* 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 de la version ZIP de votre fonction Lambda. Remplacez *`IAM_ROLE_ARN`* par le numéro de ressource Amazon (ARN) du

rôle IAM que vous avez créé dans le sujet de ce didacticiel. Créez les AWS ressources 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 :

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 Events/ 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.

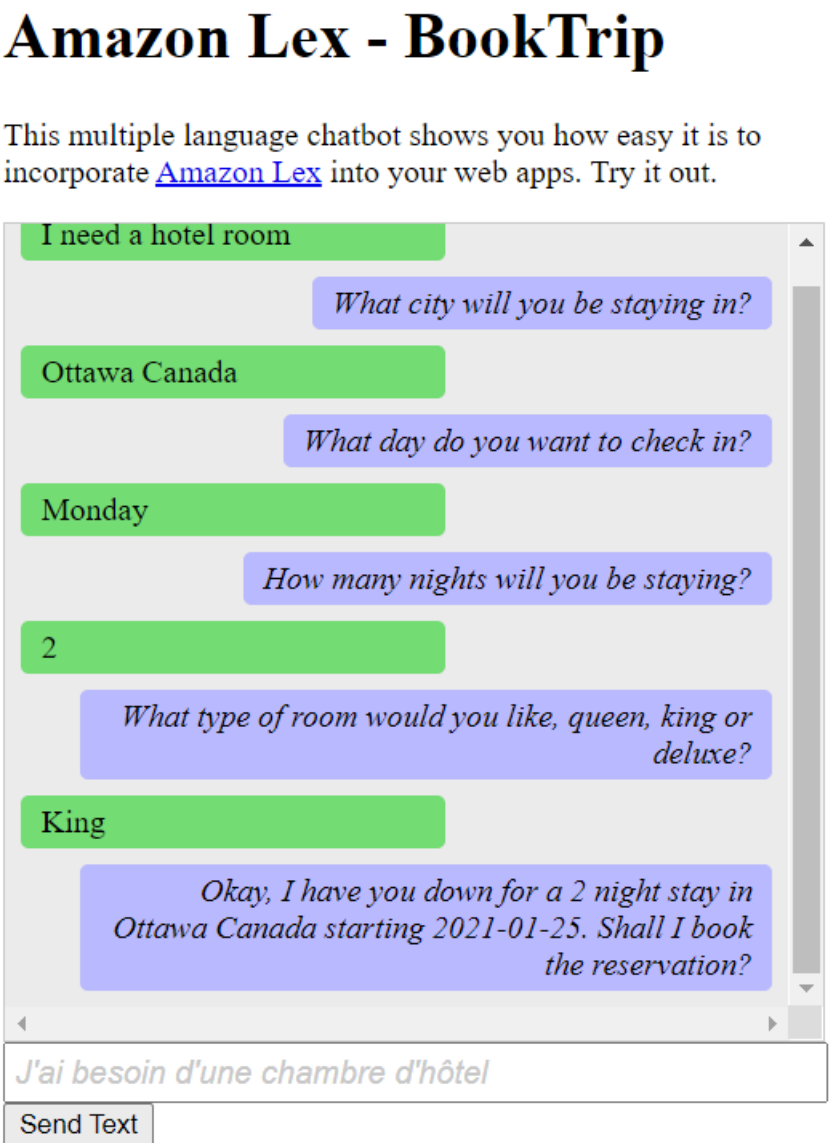
Supprimer les ressources

Félicitations ! Vous avez invoqué une fonction Lambda par le biais d'événements CloudWatch planifiés par Amazon à l'aide du. AWS SDK for 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 AWS CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

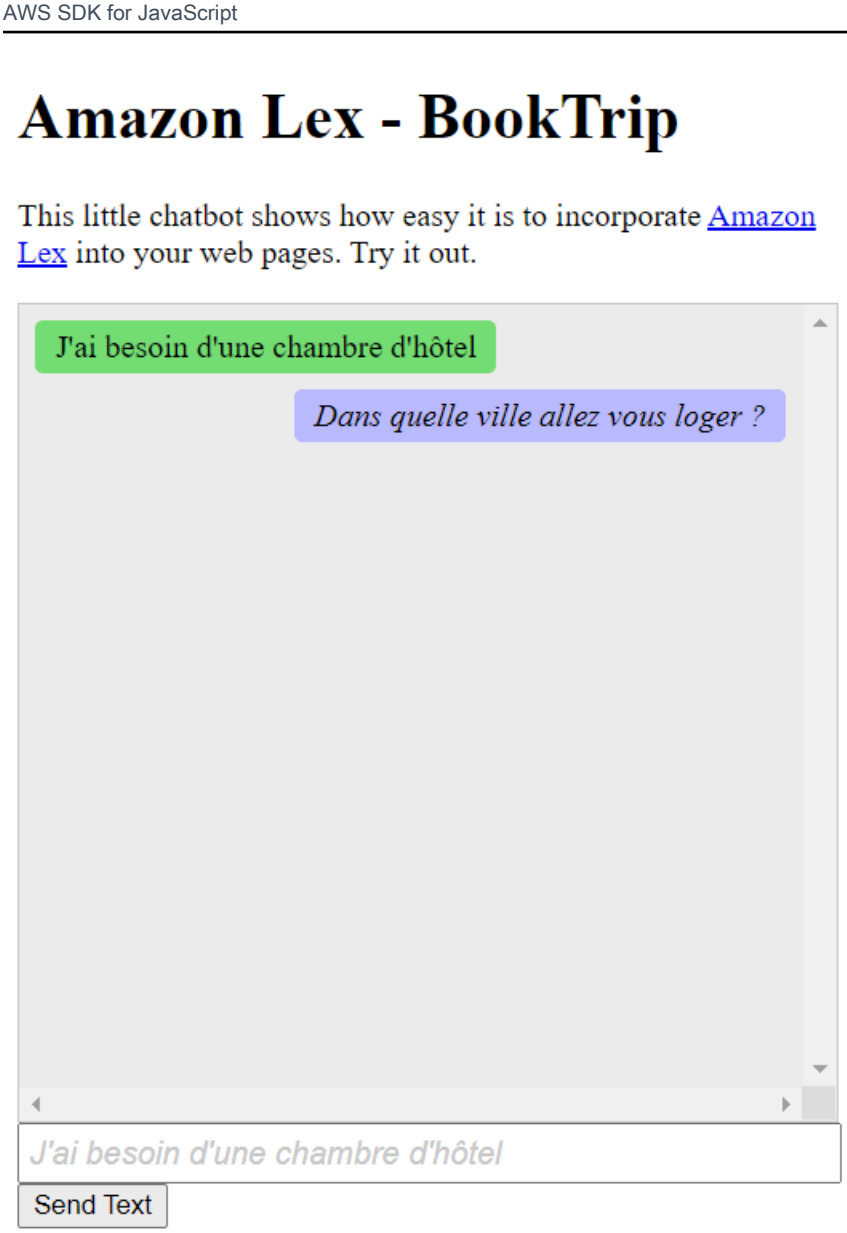
1. Ouvrez la AWS CloudFormation console.
2. Sur la page Stacks, sélectionnez la pile.
3. Sélectionnez Supprimer.

Création d'un chatbot Amazon Lex

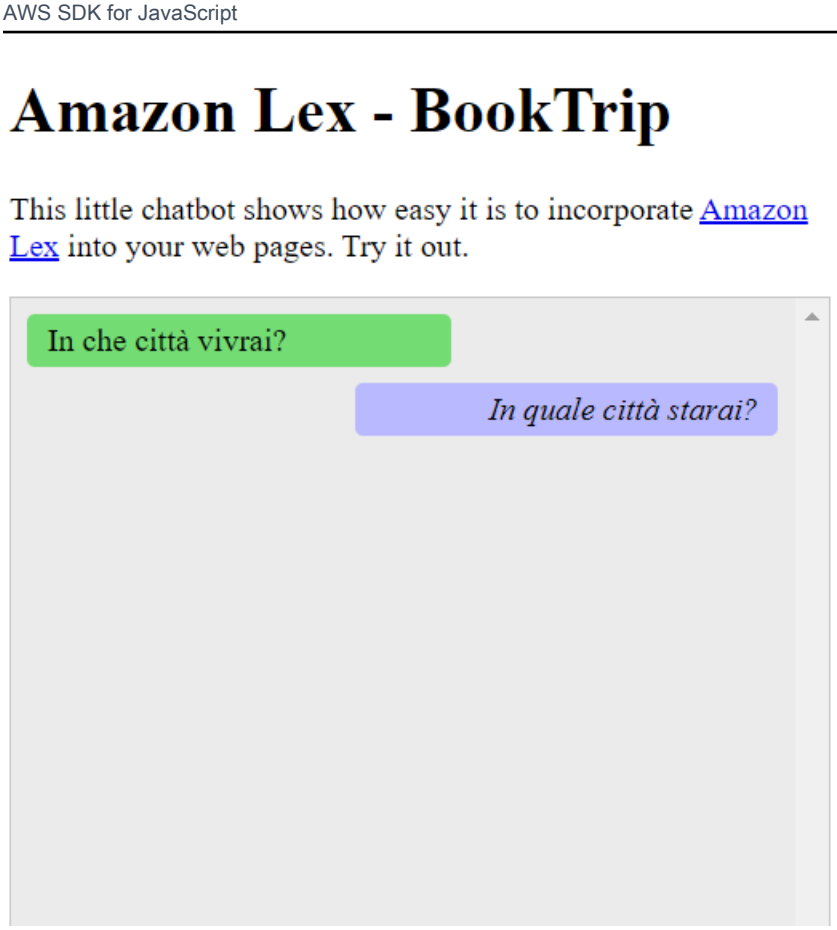
Vous pouvez créer un chatbot Amazon Lex dans une application Web pour engager les visiteurs de votre site Web. Un chatbot Amazon Lex est une fonctionnalité qui permet de discuter en ligne avec les utilisateurs sans établir de contact direct avec une personne. Par exemple, l'illustration suivante montre 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 for 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. Prérequis
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

Prérequis

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 for 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 des SDK et des outils.

> **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 Syntaxe ES6/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 AWS CloudFormation

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

Pour créer la AWS CloudFormation pile à l'aide de AWS CLI :

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

   > **Note**
   >
   > Le AWS CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour en savoir plus sur le AWS CDK, veuillez consulter le Guide du développeur AWS Cloud Development Kit (AWS CDK).

3. Exécutez la commande suivante depuis la ligne de commande, en remplaçant *`STACK_NAME 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 AWS CloudFormation l'utilisateur.

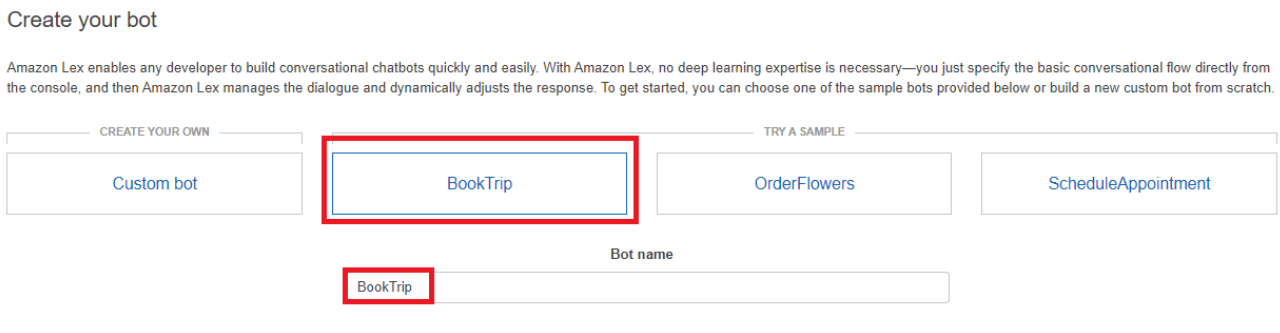
   Pour afficher les ressources créées, ouvrez la console Amazon Lex, choisissez la pile, puis sélectionnez l'onglet Ressources.

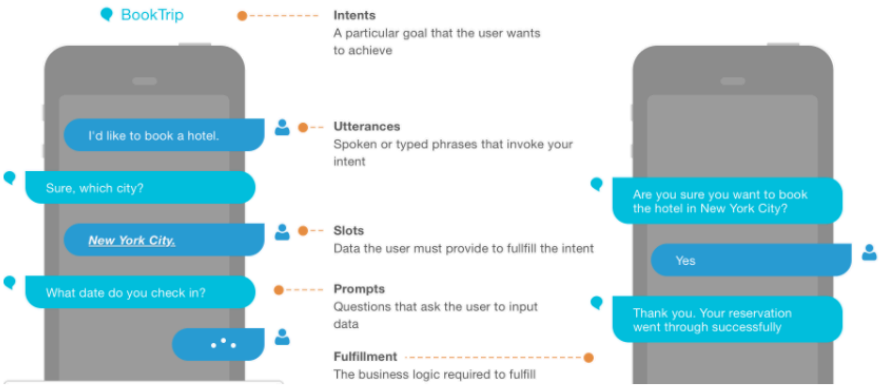
Créer un bot Amazon Lex

> **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.

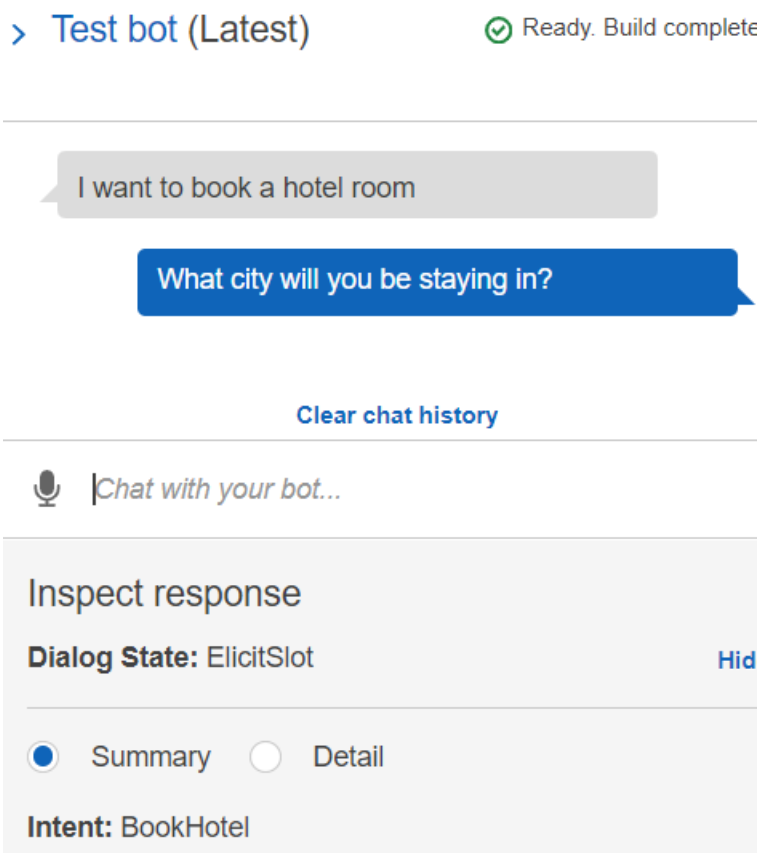
La première étape consiste à créer un chatbot Amazon Lex à l'aide de la console de gestion Amazon Web Services. Dans cet exemple, l'BookTripexemple Amazon Lex est utilisé. Pour plus d'informations, voir Book Trip.

- 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 (laissez 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 saisissant Je souhaite réserver une chambre d'hôtel.



- Choisissez Publier et spécifiez un nom d'alias (vous aurez besoin de cette valeur pour utiliser leAWS SDK for 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 for 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 for 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 le. Créez 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 chaque fichier et 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 AWS CloudFormation

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

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

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

export { comprehendClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { LexRuntimeServiceClient } from "@aws-sdk/client-lex-runtime-service";

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

// Create an Amazon Lex service client object.
const lexClient = new LexRuntimeServiceClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
```

```
  }),
});

export { lexClient };
```

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

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

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

export { translateClient };
```

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";

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

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

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

  var lexTextResponse = lexResponse;

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

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

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

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

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

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

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

Ce code est disponible ici GitHub. .

Utilisez maintenant webpack pour regrouper les AWS SDK for JavaScript modules `index.js` et dans un seul fichier,`main.js`.

1. Si ce n'est pas déjà fait, suivez cet exemple Prérequis 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 AWS CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

1. Ouvrez la AWS 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 for JavaScriptinterservices.

Création d'un exemple d'application de messagerie

Vous pouvez créer une AWS application qui envoie et récupère des messages à l'AWS SDK for JavaScriptaide d'Amazon Simple Queue Service (Amazon SQS). Les messages sont stockés dans une file d'attente « premier entré, premier sorti » (FIFO) qui garantit la cohérence de l'ordre des messages. Par exemple, le premier message stocké dans la file d'attente est le premier message lu depuis la file d'attente.

> **Note**
>
> Pour plus d'informations sur Amazon SQS, consultez Qu'est-ce qu'Amazon Simple Queue Service ?

Dans ce didacticiel, vous allez créer une application Node.js nommée AWS Messaging.

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 :

1. Prérequis
2. Allocation des ressources
3. Comprendre le flux de travail

4. Créez le code HTML
5. Créez le script du navigateur
6. Étapes suivantes

Prérequis

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 for 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 des SDK et des outils.

> **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 Syntaxe ES6/CommonJS.

Créez les AWS ressources

Ce didacticiel nécessite les ressources suivantes.

- Rôle IAM non authentifié avec des autorisations pour Amazon SQS.
- Une file d'attente Amazon SQS FIFO nommée Message.FIFO. Pour plus d'informations sur la création d'une file d'attente, consultez Création d'une file d'attente Amazon SQS.

Vous pouvez créer ces ressources manuellement, mais nous vous recommandons de les approvisionner à l'aide du AWS CloudFormation (AWS CloudFormation) comme décrit dans ce didacticiel.

> **Note**
>
> AWS CloudFormationIl s'agit d'un framework de développement logiciel qui vous permet de définir les ressources des applications cloud. Pour plus d'informations, consultez le Guide de l'utilisateur AWS CloudFormation.

Créez les AWS ressources à l'aide du AWS CloudFormation

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

Pour créer la AWS CloudFormation pile à l'aide de AWS CLI :

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

   > **Note**
   >
   > Le AWS CloudFormation modèle a été généré à l'aide du modèle AWS CDK disponible ici GitHub. Pour en savoir plus sur le AWS CDK, veuillez consulter le Guide du développeur AWS Cloud Development Kit (AWS CDK).

3. Exécutez la commande suivante depuis la ligne de commande, en remplaçant *`STACK_NAME 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 AWS CloudFormation l'utilisateur.

Pour afficher les ressources créées, ouvrez AWS CloudFormation dans la console AWS de gestion, choisissez la pile, puis sélectionnez l'onglet Ressources.

Comprendre l'application AWS de messagerie

Pour envoyer un message à une file d'attente SQS, saisissez-le dans l'application et choisissez Envoyer.

Une fois le message envoyé, l'application l'affiche.

Vous pouvez choisir Purger pour purger les messages de la file d'attente Amazon SQS. Cela entraîne une file d'attente vide et aucun message n'est affiché dans l'application.

Voici comment l'application gère un message :

- L'utilisateur sélectionne son nom, saisit son message, puis soumet le message, ce qui lance la `pushMessage` fonction.
- `pushMessage`récupère l'URL de la file d'attente Amazon SQS, puis envoie un message avec une valeur d'ID de message unique (un GUID), le texte du message et l'utilisateur à la file d'attente Amazon SQS.
- `pushMessage`extrait les messages de la file d'attente Amazon SQS, extrait l'utilisateur et le message pour chaque message, puis affiche les messages.
- L'utilisateur peut purger les messages, ce qui les supprime de la file d'attente Amazon SQS et de l'interface utilisateur.

Création de la page HTML

Vous devez maintenant créer les fichiers HTML requis pour l'interface utilisateur graphique (GUI) de l'application. 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 for JavaScript modules requis.

```
<!doctype html>
<html
  xmlns:th="http://www.thymeleaf.org"
```

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

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

      .messageelement img {
        float: left;
        max-width: 60px;
        width: 100%;
        margin-right: 20px;
        border-radius: 50%;
      }

      .messageelement img.right {
        float: right;
        margin-left: 20px;
        margin-right: 0;
```

```
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h2>AWS Sample Messaging Application</h2>
      <div id="messages"></div>

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

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

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

Ce code est également disponible ici sur GitHub.

Création du script de navigateur

Dans cette rubrique, vous allez créer un script de navigateur pour l'application. Lorsque vous avez créé le script de navigateur, vous le regroupez dans un fichier appelé `main.js` comme décrit dansRegrouper les JavaScript.

Créez un fichier nommé `index.js`. Copiez et collez le code à partir d'ici. GitHub

Ce code est expliqué dans les sections suivantes :

1. Configuration
2. Remplir le chat
3. messages push
4. purger

Configuration

Créez d'abord un `libs` répertoire et créez l'objet client Amazon SQS requis en créant un fichier nommé. `sqsClient.js` Remplacez *`REGION`* et *`IDENTITY_POOL_ID`* dans chacun d'eux.

> **Note**
>
> Utilisez l'ID du pool d'identités Amazon Cognito dans lequel vous avez créé. Créez les AWS ressources

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import {SQSClient} from "@aws-sdk/client-sqs";
const REGION = "REGION"; //e.g. "us-east-1"
const IdentityPoolId = "IDENTITY_POOL_ID";
const sqsClient = new SQSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IdentityPoolId
  }),
});
```

Dans le`index.js`, importez les AWS SDK for JavaScript modules et commandes requis. Remplacez *`SQS_QUEUE_NAME par le nom`* de la file d'attente Amazon SQS que vous avez créée dans le. Créez les AWS ressources

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

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

Remplir le chat

La `populateChat` fonction onload récupère automatiquement l'URL de la file d'attente Amazon SQS, récupère tous les messages de la file d'attente et les affiche.

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

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

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

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

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

Messages push

L'utilisateur sélectionne son nom, saisit son message, puis soumet le message, ce qui lance la `pushMessage` fonction. `pushMessage`récupère l'URL de la file d'attente Amazon SQS, puis envoie un message avec une valeur d'ID de message unique (un GUID), le texte du message et l'utilisateur à la file d'attente Amazon SQS. Il extrait ensuite tous les messages de la file d'attente Amazon SQS et les affiche.

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

  // Create random deduplication ID.
  var dt = new Date().getTime();
  var uuid = "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (
    c
  ) {
    var r = (dt + Math.random() * 16) % 16 | 0;
    dt = Math.floor(dt / 16);
    return (c == "x" ? r : (r & 0x3) | 0x8).toString(16);
  });
```

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

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

    // Retrieve messages from SQS Queue.
    const final = await sqsClient.send(
      new ReceiveMessageCommand(getMessageParams)
    );
    console.log("Successfully retrieved", final.Messages);
    $("#messages").empty();
    // Loop through messages for user and message body.
    var i;
```

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

Purger les messages

`purge`supprime les messages de la file d'attente Amazon SQS et de l'interface utilisateur.

```
// Delete the message from the Amazon SQS queue.
const purge = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
```

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

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

Regrouper les JavaScript

Ce code de script de navigateur complet est disponible ici. GitHub .

Utilisez maintenant webpack pour regrouper les AWS SDK for JavaScript modules `index.js` et dans un seul fichier,`main.js`.

1. Si ce n'est pas déjà fait, suivez cet exemple Prérequis 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 index.js --mode development --target web --devtool false -o main.js
   ```

Étapes suivantes

Félicitations ! Vous avez créé et déployé l'application AWS de messagerie qui utilise Amazon SQS. Comme indiqué au début de ce didacticiel, veillez à désactiver toutes les ressources que vous créez pendant que vous suivez ce didacticiel afin de vous assurer qu'elles ne vous seront plus facturées.

Pour ce faire, supprimez la AWS CloudFormation pile que vous avez créée dans la Créez les AWS ressources rubrique de ce didacticiel, comme suit :

1. Ouvrez le AWS CloudFormationdans la console AWS de gestion.
2. Ouvrez la page Stacks et sélectionnez la pile.
3. Sélectionnez Supprimer.

À utiliser AWS Cloud9 avec AWS SDK for JavaScript

Vous pouvez utiliser AWS Cloud9 avec le AWS SDK for JavaScript pour écrire et exécuter votre code JavaScript dans le navigateur, ainsi que pour écrire, exécuter et déboguer votre code Node.js, en utilisant simplement un navigateur. AWS Cloud9 inclut des outils tels qu'un éditeur de code et un terminal, ainsi qu'un débogueur pour le code Node.js.

L' AWS Cloud9 IDE étant basé sur le cloud, vous pouvez travailler sur vos projets depuis votre bureau, votre domicile ou n'importe où à l'aide d'une machine connectée à Internet. Pour des informations générales à ce sujet AWS Cloud9, consultez le guide de AWS Cloud9 l'utilisateur.

Les étapes suivantes décrivent la procédure de configuration à l' AWS Cloud9 aide du SDK pour JavaScript.

Table des matières

- Étape 1 : Configurez votre AWS compte pour utiliser AWS Cloud9
- Étape 2 : Configuration de votre environnement AWS Cloud9 de développement
- Étape 3 : configurer le SDK pour JavaScript
  - Pour configurer le SDK JavaScript pour Node.js
  - Pour configurer le SDK JavaScript dans le navigateur
- Étape 4 : Téléchargez un exemple de code
- Étape 5 : Exécuter et déboguer un exemple de code

Étape 1 : Configurez votre AWS compte pour utiliser AWS Cloud9

Commencez à l'utiliser en vous AWS Cloud9 connectant à la AWS Cloud9 console en tant qu'entité AWS Identity and Access Management (IAM) (par exemple, un utilisateur IAM) disposant d'autorisations d'accès pour AWS Cloud9 votre AWS compte.

Pour configurer une entité IAM dans votre AWS compte afin d'accéder AWS Cloud9à la AWS Cloud9 console et de vous y connecter, consultez la section Configuration de l'équipe AWS Cloud9 dans le guide de l'AWS Cloud9 utilisateur.

Étape 2 : Configuration de votre environnement AWS Cloud9 de développement

Une fois connecté à la AWS Cloud9 console, utilisez-la pour créer un environnement de AWS Cloud9 développement. Après avoir créé l'environnement, AWS Cloud9 ouvre l'IDE correspondant à cet environnement.

Voir Création d'un environnement AWS Cloud9 dans le guide de l'AWS Cloud9 utilisateur pour plus de détails.

> **Note**
>
> Si vous créez votre environnement dans la console pour la première fois, nous vous recommandons de choisir l'option Create a new instance for environment (EC2) (Créer une nouvelle instance pour l'environnement (EC2)). Cette option indique AWS Cloud9 de créer un environnement, de lancer une instance Amazon EC2, puis de connecter la nouvelle instance au nouvel environnement. C'est le moyen le plus rapide de commencer à l'utiliser AWS Cloud9.

Étape 3 : configurer le SDK pour JavaScript

Après avoir AWS Cloud9 ouvert l'IDE pour votre environnement de développement, suivez l'une des procédures suivantes ou les deux pour utiliser l'IDE afin de configurer le SDK pour JavaScript votre environnement.

Pour configurer le SDK JavaScript pour Node.js

1. Si le terminal n'est pas déjà ouvert dans l'IDE, ouvrez-le. Pour ce faire, choisissez Window, New Terminal (Fenêtre, Nouveau terminal) dans la barre de menus de l'IDE.
2. Exécutez la commande suivante `npm` à utiliser pour installer le `Cloud9` client du SDK pour JavaScript.

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

   Si l'IDE ne le trouve pas`npm`, exécutez les commandes suivantes, une par une, dans l'ordre suivant, pour procéder à l'installation`npm`. (Ces commandes supposent que vous avez choisi

l'option Create a new instance for environment (EC2) (Créer une nouvelle instance pour l'environnement (EC2)) plus haut dans cette rubrique.)

> **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 référentiel nvm (Node Version Manager).

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

Pour configurer le SDK JavaScript dans le navigateur

Pour utiliser le SDK JavaScript dans vos pages HTML, regroupez WebPack les modules client requis et toutes les JavaScript fonctions requises dans un seul JavaScript fichier, puis ajoutez-le dans une balise `<head>` de script dans vos pages HTML. Par exemple :

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

> **Note**
>
> Pour plus d'informations sur Webpack, voir Regroupez des applications avec Webpack

Étape 4 : Téléchargez un exemple de code

Utilisez le terminal que vous avez ouvert à l'étape précédente pour télécharger un exemple de code pour le SDK JavaScript dans l'environnement de AWS Cloud9 développement. (Si le terminal n'est pas déjà ouvert dans l'IDE, ouvrez-le en choisissant Window, New Terminal (Fenêtre, Nouveau terminal) dans la barre de menus de l'IDE.)

Pour télécharger l'exemple de code, exécutez la commande suivante : Cette commande télécharge une copie de tous les exemples de code utilisés dans la documentation officielle du AWS SDK dans le répertoire racine de votre environnement.

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

Pour trouver des exemples de code pour le SDK pour JavaScript, utilisez la fenêtre Environnement pour ouvrir le*ENVIRONMENT_NAME*`\aws-doc-sdk-examples\javascriptv3\example_code/src`, où *ENVIRONMENT_NAME est le nom* de votre AWS Cloud9 environnement de développement.

Pour savoir comment utiliser ces exemples de code et d'autres, consultez le SDK pour obtenir des exemples de JavaScript code.

Étape 5 : Exécuter et déboguer un exemple de code

Pour exécuter du code dans votre environnement de AWS Cloud9 développement, consultez la section Exécuter votre code dans le Guide de AWS Cloud9 l'utilisateur.

Pour déboguer le code Node.js, voir Déboguer votre code dans le guide de l'AWS Cloud9 utilisateur.

SDK pour exemples de JavaScript code (v3)

Les exemples de code présentés dans cette rubrique vous montrent comment utiliser le AWS SDK for JavaScript (v3) avec 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Les Exemples de services croisés sont des exemples d'applications fonctionnant sur plusieurs Services AWS.

Exemples

- Actions et scénarios utilisant le SDK pour JavaScript (v3)
- Exemples interservices utilisant le SDK pour JavaScript (v3)

Actions et scénarios utilisant le SDK pour JavaScript (v3)

Les exemples de code suivants montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK for JavaScript (v3) avec Services 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Services

- 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 pour Amazon Bedrock utilisant le SDK pour JavaScript (v3)
- Exemples d'agents pour Amazon Bedrock Runtime 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 de fournisseurs d'identité Amazon Cognito utilisant le SDK pour JavaScript (v3)
- Exemples DynamoDB utilisant le SDK JavaScript pour (v3)
- Exemples d'Amazon EC2 utilisant le SDK pour JavaScript (v3)
- Exemples d'Elastic Load Balancing utilisant le SDK pour JavaScript (v3)
- EventBridge exemples d'utilisation du 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)
- Exemples Lambda 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 Redshift utilisant le SDK pour JavaScript (v3)
- Exemples d'Amazon S3 utilisant le SDK pour JavaScript (v3)
- Exemples de S3 Glacier utilisant le SDK pour JavaScript (v3)
- SageMaker exemples d'utilisation du SDK pour JavaScript (v3)
- Exemples de Secrets Manager utilisant le SDK pour JavaScript (v3)
- Exemples Amazon SES utilisant le SDK pour JavaScript (v3)
- Exemples d'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)

- AWS Support exemples d'utilisation du SDK pour JavaScript (v3)
- Exemples d'Amazon Transcribe utilisant le SDK pour JavaScript (v3)

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions
- Scénarios

Actions

Associer un groupe cible ELB à un groupe Auto Scaling

L'exemple de code suivant montre comment associer un groupe cible ELB à un groupe Auto Scaling.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for 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 instance EC2 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 "url";

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

Créez des étapes pour déployer toutes les ressources.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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 { 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 () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    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,
        },
```

```
    }),
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
}),
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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    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"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    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,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    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("${LB_NAME}", NAMES.loadBalancerName),
    ),
    new ScenarioAction("createLoadBalancer", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const { LoadBalancers } = await client.send(
        new CreateLoadBalancerCommand({
          Name: NAMES.loadBalancerName,
          Subnets: state.subnets,
        }),
      );
      state.loadBalancerDns = LoadBalancers[0].DNSName;
      state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
      await waitUntilLoadBalancerAvailable(
        { client },
        { Names: [NAMES.loadBalancerName] },
      );
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    }),
    new ScenarioOutput("createdLoadBalancer", (state) =>
      MESSAGES.createdLoadBalancer
        .replace("${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) => {
      // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
      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 },
          ],
        }),
```

```
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
```

```
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
```

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

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
```

```
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

Créez des étapes pour exécuter la démo.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  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: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      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];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      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",
    }),
```

```
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Créez des étapes pour déployer toutes les ressources.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} 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 { 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 = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
```

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

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
```

```
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
```

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

```
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
```

```
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
```

```
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
```

```
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    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;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
```

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

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

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

```
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

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

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

```
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

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

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour Amazon Bedrock

Les exemples de code suivants montrent 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.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

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

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

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

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

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

  for (let model of models) {
    console.log("=".repeat(42));
    console.log(` Model: ${model.modelId}`);
    console.log("-".repeat(42));
    console.log(` Name: ${model.modelName}`);
    console.log(` Provider: ${model.providerName}`);
    console.log(` Model ARN: ${model.modelArn}`);
    console.log(` Input modalities: ${model.inputModalities}`);
    console.log(` Output modalities: ${model.outputModalities}`);
    console.log(` Supported customizations: ${model.customizationsSupported}`);
    console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
    console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
    console.log("=".repeat(42) + "\n");
  }

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

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

  return response;
};

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

- Pour plus de détails sur l'API, reportez-vous ListFoundationModelsà la section Référence des AWS SDK for JavaScript API.

Rubriques

- Actions

Actions

Obtenez des informations sur un modèle de fondation Amazon Bedrock

L'exemple de code suivant montre comment obtenir des informations sur un modèle de fondation 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.

Obtenez des informations sur un modèle de fondation.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

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

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 models.
```

```
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();

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

  const response = await client.send(command);

  return response.modelDetails;
};

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

- Pour plus de détails sur l'API, reportez-vous GetFoundationModelà la section Référence des AWS SDK for JavaScript API.

Liste des modèles de fondation Amazon Bedrock disponibles

L'exemple de code suivant montre comment répertorier les modèles de fondation Amazon Bedrock disponibles.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 base disponibles.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

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

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

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

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

  const command = new ListFoundationModelsCommand(input);

  const response = await client.send(command);

  return response.modelSummaries;
};

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

- Pour plus de détails sur l'API, reportez-vous ListFoundationModelsà la section Référence des AWS SDK for 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 en utilisant le AWS SDK for JavaScript (v3) avec Amazon Bedrock 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Génération de texte avec AI21 Labs Jurassic-2

L'exemple de code suivant montre comment invoquer le modèle Jurassic-2 d'AI21 Labs sur Amazon Bedrock pour générer du texte.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Invoquez le modèle de base Jurassic-2 d'AI21 Labs pour générer du texte.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

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

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

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

/**
 * Invokes the AI21 Labs Jurassic-2 large-language model to run an inference
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Jurassic-2 to complete.
 * @returns {string} The inference response (completion) from the model.
 */
export const invokeJurassic2 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "ai21.j2-mid-v1";

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for AI21 Labs Jurassic-2, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
jurassic2.html
   */
  const payload = {
    prompt,
    maxTokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
```

```
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.completions[0].data.text;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: AI21 Labs Jurassic-2");
  console.log(`Prompt: ${prompt}`);

  const completion = await invokeJurassic2(prompt);
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK for JavaScript API.

Génération de texte avec Amazon Titan Text G1

L'exemple de code suivant montre comment invoquer le modèle Amazon Titan Text G1 sur Amazon Bedrock pour générer du texte.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Invoquez le modèle de base Amazon Titan Text G1 pour générer du texte.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

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

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

/**
 * Invokes the Titan Text G1 - Express model to run an inference
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Titan Text Express to complete.
 * @returns {object[]} The inference response (results) from the model.
 */
export const invokeTitanTextExpressV1 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "amazon.titan-text-express-v1";
```

```
  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Titan text, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-titan-
text.html
   */
  const textGenerationConfig = {
    maxTokenCount: 4096,
    stopSequences: [],
    temperature: 0,
    topP: 1,
  };

  const payload = {
    inputText: prompt,
    textGenerationConfig,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);
    return responseBody.results;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = `Meeting transcript: Miguel: Hi Brant, I want to discuss the
 workstream
    for our new product launch Brant: Sure Miguel, is there anything in particular
 you want
    to discuss? Miguel: Yes, I want to talk about how users enter into the product.
    Brant: Ok, in that case let me add in Namita. Namita: Hey everyone
    Brant: Hi Namita, Miguel wants to discuss how users enter into the product.
    Miguel: its too complicated and we should remove friction.
    for example, why do I need to fill out additional forms?
    I also find it difficult to find where to access the product
    when I first land on the landing page. Brant: I would also add that
    I think there are too many steps. Namita: Ok, I can work on the
    landing page to make the product more discoverable but brant
    can you work on the additonal forms? Brant: Yes but I would need
    to work with James from another team as he needs to unblock the sign up
 workflow.
    Miguel can you document any other concerns so that I can discuss with James only
 once?
    Miguel: Sure.
    From the meeting transcript above, Create a list of action items for each
 person.`;

  console.log("\nModel: Titan Text Express v1");
  console.log(`Prompt: ${prompt}`);

  const results = await invokeTitanTextExpressV1(prompt);
  console.log("Completion:");
  for (const result of results) {
    console.log(result.outputText);
  }
  console.log("\n");
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK for JavaScript API.

Génération de texte avec Anthropic Claude 2

L'exemple de code suivant montre comment invoquer le modèle Anthropic Claude 2 sur Amazon Bedrock pour générer du texte.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Invoquez le modèle de base Anthropic Claude 2 pour générer du texte.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

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

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes the Anthropic Claude 2 model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Claude to complete.
 * @returns {string} The inference response (completion) from the model.
 */
export const invokeClaude = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "anthropic.claude-v2";

  /* Claude requires you to enclose the prompt as follows: */
  const enclosedPrompt = `Human: ${prompt}\n\nAssistant:`;

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Anthropic Claude, refer to:
```

```
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
claude.html
   */
  const payload = {
    prompt: enclosedPrompt,
    max_tokens_to_sample: 500,
    temperature: 0.5,
    stop_sequences: ["\n\nHuman:"],
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.completion;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Anthropic Claude v2");
  console.log(`Prompt: ${prompt}`);

  const completion = await invokeClaude(prompt);
```

```
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK for JavaScript API.

Génération de texte avec Meta Llama 2 Chat

L'exemple de code suivant montre comment invoquer le modèle de chat Meta Llama 2 sur Amazon Bedrock pour générer du texte.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Invoquez le modèle de base de Meta Llama 2 Chat pour générer du texte.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

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

/**
 * @typedef {Object} ResponseBody
 * @property {generation} text
 */

/**
 * Invokes the Meta Llama 2 Chat model to run an inference
```

```
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Llama-2 to complete.
 * @returns {string} The inference response (generation) from the model.
 */
export const invokeLlama2 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "meta.llama2-13b-chat-v1";

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Meta Llama 2 Chat, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-meta.html
   */
  const payload = {
    prompt,
    temperature: 0.5,
    top_p: 0.9,
    max_gen_len: 512,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.generation;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
```

```
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Meta Llama 2 Chat");
  console.log(`Prompt: ${prompt}`);

  const completion = await invokeLlama2(prompt);
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK for JavaScript API.

Génération de texte avec Mistral 7B

L'exemple de code suivant montre comment invoquer le modèle Mistral 7B sur Amazon Bedrock pour générer du texte.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Invoquez le modèle de base du Mistral 7B pour générer du texte.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
```

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

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

/**
 * Invokes the Mistral 7B model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Mistral to complete.
 * @returns {string[]} A list of inference responses (completions) from the model.
 */
export const invokeMistral7B = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-west-2" });

  const modelId = "mistral.mistral-7b-instruct-v0:2";

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

  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);
```

```
    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.outputs.map((output) => output.text);
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Mistral 7B");
  console.log(`Prompt: ${prompt}`);

  const completions = await invokeMistral7B(prompt);
  completions.forEach((completion) => {
    console.log("Completion:");
    console.log(completion);
    console.log("\n");
  });
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK for JavaScript API.

Génération de texte avec Mixtral 8x7B

L'exemple de code suivant montre comment invoquer le modèle Mixtral 8x7B sur Amazon Bedrock pour générer du texte.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Invoquez le modèle de base Mixtral 8x7B pour générer du texte.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
import { invokeMistral7B } from "./invoke-mistral7b.js";

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

/**
 * Invokes the Mixtral 8x7B model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Mistral to complete.
 * @returns {string[]} A list of inference responses (completions) from the model.
 */
export const invokeMixtral8x7B = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-west-2" });

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

```
  const modelId = "mistral.mixtral-8x7b-instruct-v0:1";

  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.outputs.map((output) => output.text);
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Mixtral 8x7B");
  console.log(`Prompt: ${prompt}`);

  const completions = await invokeMistral7B(prompt);
  completions.forEach((completion) => {
```

```
    console.log("Completion:");
    console.log(completion);
    console.log("\n");
  });
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK for JavaScript API.

Exemples d'agents pour 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 for JavaScript (v3) avec Agents for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour Agents pour Amazon Bedrock

L'exemple de code suivant montre comment commencer à utiliser Agents for 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.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

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

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

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 region.
 * It then retrieves a list of existing agents using the streamlined paginator
 approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
 * - Use of the Bedrock Agents client to initialize and communicate with the AWS
 service.
 * - Listing resources in a paginated response pattern.
 * - Accessing an individual resource using a command object.
 *
 * @returns {Promise<void>} A promise that resolves when the function has completed
 execution.
 */
export const main = async () => {
  const region = "us-east-1";

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

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

  console.log(`Retrieving the list of existing agents...`);
```

```
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});

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

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

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

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

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

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - GetAgent
  - ListAgents

Rubriques

- Actions

Actions

Création d'un agent

L'exemple de code suivant montre comment créer un agent 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.

Créez un agent .

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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

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

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

```
export const createAgent = async (
  agentName,
  foundationModel,
  agentResourceRoleArn,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

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

  return response.agent;
};

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

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

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

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

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

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

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

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

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

- Pour plus de détails sur l'API, reportez-vous CreateAgentà la section Référence des AWS SDK for JavaScript API.

Suppression d’un agent

L'exemple de code suivant montre comment supprimer un agent 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.

Supprimez un agent.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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

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

/**
 * Deletes an Amazon Bedrock Agent.
```

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

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

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

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

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

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

- Pour plus de détails sur l'API, reportez-vous DeleteAgentà la section Référence des AWS SDK for JavaScript API.

Obtention d’informations sur un agent

L'exemple de code suivant montre comment obtenir des informations sur un agent 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.

Trouvez un agent.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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

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

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

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

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

  // The agentId must be an alphanumeric string with exactly 10 characters.
```

```
  const agentId = "[ABC123DE45]";

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

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

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

- Pour plus de détails sur l'API, reportez-vous GetAgentà la section Référence des AWS SDK for JavaScript API.

Répertorier les groupes d'actions d'un agent

L'exemple de code suivant montre comment répertorier les groupes d'actions pour un agent 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.

Répertoriez les groupes d'actions d'un agent.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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

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

```
/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  // Create a paginator configuration
  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const params = { agentId, agentVersion };

  const pages = paginateListAgentActionGroups(paginatorConfig, params);

  // Paginate until there are no more results
  const actionGroupSummaries = [];
  for await (const page of pages) {
    actionGroupSummaries.push(...page.actionGroupSummaries);
  }

  return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
```

```
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const actionGroupSummaries = [];
  do {
    const command = new ListAgentActionGroupsCommand({
      agentId,
      agentVersion,
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
    const response = await client.send(command);

    for (const actionGroup of response.actionGroupSummaries || []) {
      actionGroupSummaries.push(actionGroup);
    }

    nextToken = response.nextToken;
  } while (nextToken);

  return actionGroupSummaries;
};

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

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

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
 'DRAFT').
  const agentVersion = "[DRAFT]";

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

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using ListAgentActionGroupsCommand:",
  );

  for (const actionGroup of await listAgentActionGroupsWithCommandObject(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using the paginateListAgents function:",
  );
  for (const actionGroup of await listAgentActionGroupsWithPaginator(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListAgentActionGroupsà la section Référence des AWS SDK for JavaScript API.

Répertorier les agents disponibles

L'exemple de code suivant montre comment répertorier les agents d'Amazon Bedrock associés à un compte.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

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

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
```

```
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 ListAgentsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const agentSummaries = [];
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
    const paginatedResponse = await client.send(command);

    agentSummaries.push(...(paginatedResponse.agentSummaries || []));

    nextToken = paginatedResponse.nextToken;
  } while (nextToken);

  return agentSummaries;
};

// Invoke main function if this file was run directly.
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
  console.log("Listing agents using ListAgentsCommand:");
  for (const agent of await listAgentsWithCommandObject()) {
    console.log(agent);
  }

  console.log("=".repeat(68));
  console.log("Listing agents using the paginateListAgents function:");
  for (const agent of await listAgentsWithPaginator()) {
    console.log(agent);
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListAgentsà la section Référence des AWS SDK for JavaScript API.

Exemples d'agents pour Amazon Bedrock Runtime 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 for JavaScript (v3) avec des agents pour Amazon Bedrock 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Invoquer un agent

L'exemple de code suivant montre comment invoquer un agent 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.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  BedrockAgentRuntimeClient,
  InvokeAgentCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
```

```
  // });

  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (let chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
      console.log(chunk);
      const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
      completion += decodedResponse;
    }

    return { sessionId: sessionId, completion };
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeAgentà la section Référence des AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Créer une alerte de métrique

L'exemple de code suivant montre comment créer ou mettre à jour une CloudWatch alarme Amazon et comment l'associer à la métrique spécifiée, à l'expression mathématique métrique, au modèle de détection des anomalies ou à la requête Metrics Insights spécifiée.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.

- Pour plus de détails sur l'API, reportez-vous PutMetricAlarmà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutMetricAlarmà la section Référence des AWS SDK for JavaScript API.

Supprimer des alertes

L'exemple de code suivant montre comment supprimer les CloudWatch alarmes Amazon.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteAlarmsà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous DeleteAlarmsà la section Référence des AWS SDK for JavaScript API.

Décrire des alertes pour une métrique

L'exemple de code suivant montre comment décrire les CloudWatch alarmes Amazon pour une métrique.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeAlarmsForMetricà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeAlarmsForMetricà la section Référence des AWS SDK for JavaScript API.

Désactiver des actions d'alerte

L'exemple de code suivant montre comment désactiver les actions CloudWatch d'alarme Amazon.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.

- Pour plus de détails sur l'API, reportez-vous DisableAlarmActionsà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Pour de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous DisableAlarmActionsà la section Référence des AWS SDK for JavaScript API.

Activer des actions d'alerte

L'exemple de code suivant montre comment activer les actions CloudWatch d'alarme Amazon.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous EnableAlarmActionsà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

- Pour de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous EnableAlarmActionsà la section Référence des AWS SDK for JavaScript API.

Répertorier les métriques

L'exemple de code suivant montre comment répertorier les métadonnées des CloudWatch métriques Amazon. Pour obtenir des données pour une métrique, utilisez les GetMetricStatistics actions GetMetricData ou.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = () => {
  // Use the AWS console to see available namespaces and metric names. Custom
 metrics can also be created.
```

```
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
    Dimensions: [
      {
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

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

Créez le client dans un module séparé et exportez-le.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListMetricsà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
```

```
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListMetricsà la section Référence des AWS SDK for JavaScript API.

Placer des données dans une métrique

L'exemple de code suivant montre comment publier des points de données métriques sur Amazon CloudWatch.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.

- Pour plus de détails sur l'API, reportez-vous PutMetricDataà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
```

```
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutMetricDataà la section Référence des AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Ajout d'une cible

L'exemple de code suivant montre comment ajouter une cible à un événement Amazon CloudWatch Events.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutTargetsà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myCloudWatchEventsTarget",
    },
  ],
};

cwevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutTargetsà la section Référence des AWS SDK for JavaScript API.

Créer une règle planifiée

L'exemple de code suivant montre comment créer une règle de planification Amazon CloudWatch Events.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutRuleà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

cwevents.putRule(params, function (err, data) {
```

```
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutRuleà la section Référence des AWS SDK for JavaScript API.

Envoyez des événements

L'exemple de code suivant montre comment envoyer des CloudWatch événements Amazon Events.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutEventsà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });
```

```
var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

cwevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutEventsà la section Référence des AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions
- Scénarios

Actions

Création d'un groupe de journaux

L'exemple de code suivant montre comment créer un nouveau groupe de CloudWatch journaux Logs.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Créer un filtre d'abonnement

L'exemple de code suivant montre comment créer un filtre d'abonnement Amazon CloudWatch Logs.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutSubscriptionFilterà la section Référence des AWS SDK for JavaScript API.

Supprimer un groupe de journaux

L'exemple de code suivant montre comment supprimer un groupe de CloudWatch journaux Logs existant.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer un filtre d'abonnement

L'exemple de code suivant montre comment supprimer un filtre d'abonnement Amazon CloudWatch Logs.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteSubscriptionFilterà la section Référence des AWS SDK for JavaScript API.

Décrire les filtres d'abonnement existants

L'exemple de code suivant montre comment décrire les filtres d'abonnement existants d'Amazon CloudWatch Logs.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};
```

```
cwl.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeSubscriptionFiltersà la section Référence des AWS SDK for JavaScript API.

Décrire des groupes de journaux

L'exemple de code suivant montre comment décrire les groupes de CloudWatch journaux Logs.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 && 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 for JavaScript API.

Obtenir les résultats d'une requête

L'exemple de code suivant montre comment obtenir les résultats d'une requête.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Démarrage d'une session Live Tail

L'exemple de code suivant montre comment démarrer une session Live Tail pour un groupe de journaux/un flux de journaux existant.

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 la 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 for JavaScript API.

Démarrer une requête

L'exemple de code suivant montre comment démarrer une requête.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Scénarios

Exécuter 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.

C'est le 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(parseInt(process.env.QUERY_START_DATE)),
    new Date(parseInt(process.env.QUERY_END_DATE)),
  ],
});

await cloudWatchQuery.run();

console.log(
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found:
 ${cloudWatchQuery.results.length}`,
);
```

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]);
}

// snippet-start:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
// snippet-end:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]

/**
 * Starts a query and waits for it to complete.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 */
async _query(dateRange, maxLogs) {
  try {
    const { queryId } = await this._startQuery(dateRange, maxLogs);
    const { results } = await this._waitUntilQueryDone(queryId);
    return results ?? [];
  } catch (err) {
    /**
     * This error is thrown when StartQuery returns an error indicating
     * that the query's start or end date occur before the log group was
     * created.
     */
    if (err instanceof DateOutOfBoundsError) {
      return [];
    } else {
      throw err;
    }
  }
}

// snippet-start:[javascript.v3.cloudwatch-logs.actions.StartQuery]
/**
```

```
   * 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;
    }
  }
  // snippet-end:[javascript.v3.cloudwatch-logs.actions.StartQuery]

  /**
   * 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 d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - GetQueryResults
  - StartQuery

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Création d'un projet

L'exemple de code suivant montre comment créer un CodeBuild projet.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateProjectà la section Référence des AWS SDK for JavaScript API.

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour Amazon Cognito

Les exemples de code suivants montrent comment bien démarrer 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 for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Confirmation d’un utilisateur

L'exemple de code suivant montre comment confirmer un utilisateur 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.

```
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 for JavaScript API.

Confirmation d’un appareil MFA pour le suivi

L'exemple de code suivant montre comment confirmer le suivi d'un appareil MFA par 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.

```
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 for JavaScript API.

Obtention d'un jeton pour associer une application MFA à un utilisateur

L'exemple de code suivant montre comment obtenir un jeton pour associer une application MFA à un utilisateur 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.

```
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 for JavaScript API.

Obtention d'informations sur un utilisateur

L'exemple de code suivant montre comment obtenir des informations sur un utilisateur 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.

```
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 for JavaScript API.

Répertorier des utilisateurs

L'exemple de code suivant montre comment répertorier les utilisateurs d'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.

```
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 for JavaScript API.

Renvoi d'un code de confirmation

L'exemple de code suivant montre comment renvoyer un code de confirmation 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.

```
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 for JavaScript API.

Réponse aux stimulations d'authentification SRP

L'exemple de code suivant montre comment répondre aux défis d'authentification Amazon Cognito SRP.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Réponse à une stimulation d'authentification

L'exemple de code suivant montre comment répondre à un défi d'authentification 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.

```
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 for JavaScript API.

Inscription d'un utilisateur

L'exemple de code suivant montre comment inscrire un utilisateur à 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.

```
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 for JavaScript API.

Démarrer l'authentification

L'exemple de code suivant montre comment démarrer l'authentification 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.

```
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 for JavaScript API.

Démarrage de l’authentification avec les informations d’identification de l’administrateur

L'exemple de code suivant montre comment démarrer l'authentification avec Amazon Cognito et les informations d'identification de l'administrateur.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Vérification d'une application MFA avec un utilisateur

L'exemple de code suivant montre comment vérifier une application MFA auprès d'un utilisateur 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.

```
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 for JavaScript API.

Scénarios

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 { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up'
 command.`,
```

```
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Signing up.`);
    await signUp({ clientId, username, password, email });
    log(`Signed up. A confirmation email has been sent to: ${email}.`);
    log(`Run 'confirm-sign-up ${username} <code>' to confirm your account.`);
  } catch (err) {
    log(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";
```

```
const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-
sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Confirming user.`);
    await confirmSignUp({ clientId, username, code });
    log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
```

```
    );
  } catch (err) {
    log(err);
  }
};

export { confirmSignUpHandler };

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

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

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../../actions/associate-software-
token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-
token' to finish the setup.",
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};
```

```
const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-initiate-
auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);

    log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
```

```
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    log(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../actions/admin-respond-to-auth-
challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-
respond-to-auth-challenge' command.`,
    );
```

```
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an
 argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    log("Successfully authenticated.");
  } catch (err) {
    log(err);
  }
};

export { adminRespondToAuthChallengeHandler };
```

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

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-
software-token' command.`,
    );
  }
};
const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;

  try {
    validateTotp(totp);

    log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
```

```
    console.log(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

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

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge

- SignUp
- VerifySoftwareToken

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 for JavaScript (v3) avec DynamoDB.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Hello DynamoDB

Les exemples de code suivants montrent 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.

```
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 for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Créer une table

L'exemple de code suivant montre comment créer une table 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.

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

const client = new DynamoDBClient({});

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

```
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

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

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
```

```
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK for JavaScript API.

Supprimer une table

L'exemple de code suivant montre comment supprimer une table 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.

```
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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK for JavaScript API.

Supprimer un élément d'une table

L'exemple de code suivant montre comment supprimer un élément d'une table 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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 élément d'une table .

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Supprimez un élément d'une table à l'aide du client de document DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
```

```
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK for JavaScript API.

Obtenir un lot d'éléments

L'exemple de code suivant montre comment obtenir un lot d'éléments 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
```

```
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous BatchGetItemà la section Référence des AWS SDK for JavaScript API.

Obtenir un élément d'une table

L'exemple de code suivant montre comment obtenir un élément d'une table 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.

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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenir un élément d'une table

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};
```

```
// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Obtenez un élément d'une table à l'aide du client de document DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK for JavaScript API.

Obtenir des informations sur une table

L'exemple de code suivant montre comment obtenir des informations sur une table 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.

```
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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DescribeTableà la section Référence des AWS SDK for JavaScript API.

Répertorier des tables

L'exemple de code suivant montre comment répertorier les tables 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.

```
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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK for JavaScript API.

Insérer un élément dans une table

L'exemple de code suivant montre comment placer un élément dans une table 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.

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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Insérez un élément dans la table.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Insérez un élément dans une table à l'aide du client de document DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK for JavaScript API.

Interroger une table

L'exemple de code suivant montre comment interroger une table 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for JavaScript .

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK for JavaScript .

Exécuter une instruction PartiQL

L'exemple de code suivant montre comment exécuter une instruction PartiQL sur une table 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.

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 for JavaScript API.

Exécuter des lots d'instructions PartiQL

L'exemple de code suivant montre comment exécuter des lots d'instructions PartiQL sur une table 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.

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 for JavaScript API.

Analyser une table

L'exemple de code suivant montre comment scanner une table 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.

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 de plus amples informations sur API, consultez TagResource dans AWS SDK for JavaScript Référence de l'API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
```

```
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour de plus amples informations sur l'API, consultez Analyser dans la référence d'API AWS SDK for JavaScript .

Mettre à jour un élément existant dans une table

L'exemple de code suivant montre comment mettre à jour un élément dans une table 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.

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 for JavaScript API.

Écrire un lot d'éléments

L'exemple de code suivant montre comment écrire un lot d'éléments 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.

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 "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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
```

```
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK for JavaScript API.

Scénarios

Commencer à utiliser des tables, des éléments et des requêtes

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 "fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
```

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

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

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });

  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
```

```
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

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

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
```

```
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
   */

  log("Deleting a single movie from the table.");
  const deleteCommand = new DeleteCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
  });
  await client.send(deleteCommand);
  log("Movie deleted.");
```

```
  /**
   * Upload a batch of movies.
   */

  log("Adding movies from local JSON file.");
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );
  const movies = JSON.parse(file.toString());
  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        [tableName]: putRequests,
      },
    });

    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
```

```
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log(`Scan for movies released between 1980 and 1990`);
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression. Scan
 will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
```

```
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Interrogation
  - Analyser
  - UpdateItem

Interroger 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,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

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

export const main = async () => {
  /**
   * 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 for JavaScript API.

Interroger 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,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

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

```
const docClient = DynamoDBDocumentClient.from(client);

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

export const main = async () => {
  /**
   * 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 for JavaScript API.

Exemples d'Amazon EC2 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 for JavaScript (v3) avec Amazon EC2.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Hello Amazon EC2

Les exemples de code suivants montrent comment démarrer avec 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 } from "@aws-sdk/client-ec2";

import { client } from "./libs/client.js";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  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);
  }
```

```
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeSecurityGroupsà la section Référence des AWS SDK for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Allouer une adresse IP Elastic

L'exemple de code suivant montre comment allouer une adresse IP élastique pour 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 { AllocateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  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 (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous AllocateAddressà la section Référence des AWS SDK for JavaScript API.

Associer une adresse IP Elastic à une instance

L'exemple de code suivant montre comment associer une adresse IP élastique à une instance 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 { AssociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  // You need to allocate an Elastic IP address before associating it with an
 instance.
  // You can do that with the AllocateAddressCommand.
  const allocationId = "ALLOCATION_ID";
  // You need to create an EC2 instance before an IP address can be associated with
 it.
  // You can do that with the RunInstancesCommand.
  const instanceId = "INSTANCE_ID";
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    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 (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous AssociateAddressà la section Référence des AWS SDK for JavaScript API.

Création d’un modèle de lancement

L'exemple de code suivant montre comment créer un modèle de lancement 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.

```
    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 for JavaScript API.

Création d’un groupe de sécurité

L'exemple de code suivant montre comment créer un groupe de sécurité 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 { CreateSecurityGroupCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: "SECURITY_GROUP_NAME",
    // Up to 255 characters in length.
    Description: "DESCRIPTION",
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (err) {
    console.error(err);
  }
```

```
};
```

- Pour plus de détails sur l'API, reportez-vous CreateSecurityGroupà la section Référence des AWS SDK for JavaScript API.

Créer une paire de clés de sécurité

L'exemple de code suivant montre comment créer une paire de clés de sécurité pour 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 { CreateKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a key pair in Amazon EC2.
    const { KeyMaterial, KeyName } = await client.send(
      // A unique name for the key pair. Up to 255 ASCII characters.
      new CreateKeyPairCommand({ KeyName: "KEY_PAIR_NAME" }),
    );
    // This logs your private key. Be sure to save it.
    console.log(KeyName);
    console.log(KeyMaterial);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous CreateKeyPairà la section Référence des AWS SDK for JavaScript API.

Créer et exécuter une instance

L'exemple de code suivant montre comment créer et exécuter une instance 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 { RunInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Create a new EC2 instance.
export const main = async () => {
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: "KEY_PAIR_NAME",
    // Your security group.
    SecurityGroupIds: ["SECURITY_GROUP_ID"],
    // An x86_64 compatible image.
    ImageId: "ami-0001a0d1a04bfcc30",
    // An x86_64 compatible free-tier instance type.
    InstanceType: "t1.micro",
    // Ensure only 1 instance launches.
    MinCount: 1,
    MaxCount: 1,
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous RunInstancesà la section Référence des AWS SDK for JavaScript API.

Supprimer un modèle de lancement

L'exemple de code suivant montre comment supprimer un modèle de lancement 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.

```
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
```

- Pour plus de détails sur l'API, reportez-vous DeleteLaunchTemplateà la section Référence des AWS SDK for JavaScript API.

Supprimer un groupe de sécurité

L'exemple de code suivant montre comment supprimer un groupe de sécurité 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 { DeleteSecurityGroupCommand } from "@aws-sdk/client-ec2";
```

```
import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DeleteSecurityGroupCommand({
    GroupId: "GROUP_ID",
  });

  try {
    await client.send(command);
    console.log("Security group deleted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteSecurityGroupà la section Référence des AWS SDK for JavaScript API.

Supprimer une paire de clés de sécurité

L'exemple de code suivant montre comment supprimer une paire de clés de sécurité 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 { DeleteKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DeleteKeyPairCommand({
    KeyName: "KEY_PAIR_NAME",
  });

  try {
    await client.send(command);
```

```
    console.log("Successfully deleted key pair.");
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteKeyPairà la section Référence des AWS SDK for JavaScript API.

Décrire des régions

L'exemple de code suivant montre comment décrire les régions 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 { DescribeRegionsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions true even the regions that require opt-in will be returned.
    AllRegions: true,
    // You can omit the Filters property if you want to get all regions.
    Filters: [
      {
        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: ["ap-southeast-4"],
      },
    ],
  });
```

```
  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeRegionsà la section Référence des AWS SDK for JavaScript API.

Décrire des instances

L'exemple de code suivant montre comment décrire les instances 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 { DescribeInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List all of your EC2 instances running with x86_64 architecture that were
// launched this month.
export const main = async () => {
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;
  const command = new DescribeInstancesCommand({
    Filters: [
      { Name: "architecture", Values: ["x86_64"] },
      { Name: "instance-state-name", Values: ["running"] },
```

```
      {
        Name: "launch-time",
        Values: [launchTimePattern],
      },
    ],
  });

  try {
    const { Reservations } = await client.send(command);
    const instanceList = Reservations.reduce((prev, current) => {
      return prev.concat(current.Instances);
    }, []);

    console.log(instanceList);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeInstancesà la section Référence des AWS SDK for JavaScript API.

Désactiver la surveillance détaillée

L'exemple de code suivant montre comment désactiver la surveillance détaillée sur une instance 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 { UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

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

```
  const command = new UnmonitorInstancesCommand({
    InstanceIds: ["i-09a3dfe7ae00e853f"],
  });

  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 (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous UnmonitorInstancesà la section Référence des AWS SDK for JavaScript API.

Dissocier une adresse IP Elastic d’une instance

L'exemple de code suivant montre comment dissocier une adresse IP élastique d'une instance 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 { DisassociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Disassociate an Elastic IP address from an instance.
export const main = async () => {
```

```
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AssociationId: "ASSOCIATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DisassociateAddressà la section Référence des AWS SDK for JavaScript API.

Activer la surveillance

L'exemple de code suivant montre comment activer la surveillance pour une instance Amazon EC2 en cours d'exécution.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 { MonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// 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.
export const main = async () => {
  const command = new MonitorInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
```

```
  });

  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 (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous MonitorInstancesà la section Référence des AWS SDK for JavaScript API.

Obtenir des données sur des Amazon Machine Images

L'exemple de code suivant montre comment obtenir des données sur Amazon Machine Images (AMI).

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 { paginateDescribeImages } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first i386 image available for EC2 instances.
export const main = async () => {
  // 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: 25 },
    {
      // 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: ["x86_64"] }],
    },
  );

  try {
    const arm64Images = [];
    for await (const page of paginator) {
      if (page.Images.length) {
        arm64Images.push(...page.Images);
        // Once we have at least 1 result, we can stop.
        if (arm64Images.length >= 1) {
          break;
        }
      }
    }
    console.log(arm64Images);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeImagesà la section Référence des AWS SDK for JavaScript API.

Obtenir des données sur un groupe de sécurité

L'exemple de code suivant montre comment obtenir des données sur un groupe de sécurité 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 } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Log the details of a specific security group.
export const main = async () => {
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: ["SECURITY_GROUP_ID"],
  });

  try {
    const { SecurityGroups } = await client.send(command);
    console.log(JSON.stringify(SecurityGroups, null, 2));
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeSecurityGroupsà la section Référence des AWS SDK for JavaScript API.

Obtenir des données sur les types d'instances

L'exemple de code suivant montre comment obtenir des données sur les types d'instances 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 {
  paginateDescribeInstanceTypes,
  DescribeInstanceTypesCommand,
} from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first arm64 EC2 instance type available.
export const main = async () => {
  // 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: 25 },
    {
      Filters: [
        { Name: "processor-info.supported-architecture", Values: ["x86_64"] },
        { Name: "free-tier-eligible", Values: ["true"] },
      ],
    }
  );

  try {
    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(instanceTypes);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeInstanceTypesà la section Référence des AWS SDK for JavaScript API.

Obtenir des données sur le profil d'instance associé à une instance

L'exemple de code suivant montre comment obtenir des données sur le profil d'instance associé à une instance 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.

```
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 for JavaScript API.

Obtenir des informations sur les adresses IP Elastic

L'exemple de code suivant montre comment obtenir des informations sur les adresses IP Elastic.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: ["ALLOCATION_ID"],
  });

  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 (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeAddressesà la section Référence des AWS SDK for JavaScript API.

Obtenir le VPC par défaut

L'exemple de code suivant montre comment obtenir le VPC par défaut du compte actuel.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir les sous-réseaux par défaut d'un VPC

L'exemple de code suivant montre comment obtenir les sous-réseaux d'un VPC par défaut.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Répertorier les paires de clés de sécurité

L'exemple de code suivant montre comment répertorier les paires de clés de sécurité 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 { DescribeKeyPairsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

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

  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 (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeKeyPairsà la section Référence des AWS SDK for JavaScript API.

Redémarrer une instance

L'exemple de code suivant montre comment redémarrer une instance 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 { RebootInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new RebootInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous RebootInstancesà la section Référence des AWS SDK for JavaScript API.

Libérer une adresse IP Elastic

L'exemple de code suivant montre comment libérer une adresse IP élastique.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AllocationId: "ALLOCATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully released address.");
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous ReleaseAddressà la section Référence des AWS SDK for JavaScript API.

Remplacer le profil d'instance associé à une instance

L'exemple de code suivant montre comment remplacer le profil d'instance associé à une instance 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.

```
    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 for JavaScript API.

Définir des règles entrantes pour un groupe de sécurité

L'exemple de code suivant montre comment définir des règles de trafic entrant pour un groupe de sécurité 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 { AuthorizeSecurityGroupIngressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Grant permissions for a single IP address to ssh into instances
// within the provided security group.
```

```
export const main = async () => {
  const command = new AuthorizeSecurityGroupIngressCommand({
    // Replace with a security group ID from the AWS console or
    // the DescribeSecurityGroupsCommand.
    GroupId: "SECURITY_GROUP_ID",
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        // Replace 0.0.0.0 with 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: "0.0.0.0/32" }],
      },
    ],
  });

  try {
    const { SecurityGroupRules } = await client.send(command);
    console.log(JSON.stringify(SecurityGroupRules, null, 2));
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous AuthorizeSecurityGroupIngressà la section Référence des AWS SDK for JavaScript API.

Démarrer une instance

L'exemple de code suivant montre comment démarrer une instance 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 { StartInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new StartInstancesCommand({
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous StartInstancesà la section Référence des AWS SDK for JavaScript API.

Arrêter une instance

L'exemple de code suivant montre comment arrêter une instance 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 { StopInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";
```

```
export const main = async () => {
  const command = new StopInstancesCommand({
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous StopInstancesà la section Référence des AWS SDK for JavaScript API.

Résilier une instance

L'exemple de code suivant montre comment mettre fin à une instance 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 { TerminateInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new TerminateInstancesCommand({
```

```
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, reportez-vous TerminateInstancesà la section Référence des AWS SDK for 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 instance EC2 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 "url";

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

Créez des étapes pour déployer toutes les ressources.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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 { 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 () => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  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,
        },
      }),
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    );
  }),
  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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
  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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
  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"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    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,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    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] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    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 },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
```

```
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
```

```
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
```

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

Créez des étapes pour exécuter la démo.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  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: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      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];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      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",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Créez des étapes pour déployer toutes les ressources.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
```

```
} 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 { 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 = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
```

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

```
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

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

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

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

```
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
```

```
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      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.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
```

```
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    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;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
```

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

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

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

```
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

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

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

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

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
```

```
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs

- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Démarrer avec les instances

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.

Exécutez un scénario interactif à une invite de commande.

```
import { mkdtempSync, writeFileSync, rmSync } from "fs";
import { tmpdir } from "os";
import { join } from "path";
import { get } from "http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
```

```
  DeleteSecurityGroupCommand,
  DescribeInstancesCommand,
  DescribeKeyPairsCommand,
  DescribeSecurityGroupsCommand,
  DisassociateAddressCommand,
  EC2Client,
  paginateDescribeImages,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";
import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

const ec2Client = new EC2Client();
const ssmClient = new SSMClient();

const prompter = new Prompter();
const confirmMessage = "Continue?";
const tmpDirectory = mkdtempSync(join(tmpdir(), "ec2-scenario-tmp"));

const createKeyPair = async (keyPairName) => {
  // Create a key pair in Amazon EC2.
  const { KeyMaterial, KeyPairId } = await ec2Client.send(
    // A unique name for the key pair. Up to 255 ASCII characters.
    new CreateKeyPairCommand({ KeyName: keyPairName }),
  );

  // Save the private key in a temporary location.
  writeFileSync(`${tmpDirectory}/${keyPairName}.pem`, KeyMaterial, {
    mode: 0o400,
  });

  return KeyPairId;
};
```

```
const describeKeyPair = async (keyPairName) => {
  const command = new DescribeKeyPairsCommand({
    KeyNames: [keyPairName],
  });
  const { KeyPairs } = await ec2Client.send(command);
  return KeyPairs[0];
};

const createSecurityGroup = async (securityGroupName) => {
  const command = new CreateSecurityGroupCommand({
    GroupName: securityGroupName,
    Description: "A security group for the Amazon EC2 example.",
  });
  const { GroupId } = await ec2Client.send(command);
  return GroupId;
};

const allocateIpAddress = async () => {
  const command = new AllocateAddressCommand({});
  const { PublicIp, AllocationId } = await ec2Client.send(command);
  return { PublicIp, AllocationId };
};

const getLocalIpAddress = () => {
  return 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);
    });
  });
};

const authorizeSecurityGroupIngress = async (securityGroupId) => {
  const ipAddress = await getLocalIpAddress();
  const command = new AuthorizeSecurityGroupIngressCommand({
    GroupId: securityGroupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
```

```
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  await ec2Client.send(command);
  return ipAddress;
};

const describeSecurityGroup = async (securityGroupName) => {
  const command = new DescribeSecurityGroupsCommand({
    GroupNames: [securityGroupName],
  });
  const { SecurityGroups } = await ec2Client.send(command);

  return SecurityGroups[0];
};

const getAmznLinux2AMIs = async () => {
  const AMIs = [];
  for await (const page of paginateGetParametersByPath(
    {
      client: ssmClient,
    },
    { Path: "/aws/service/ami-amazon-linux-latest" },
  )) {
    page.Parameters.forEach((param) => {
      if (param.Name.includes("amzn2")) {
        AMIs.push(param.Value);
      }
    });
  }

  const imageDetails = [];

  for await (const page of paginateDescribeImages(
    { client: ec2Client },
    { ImageIds: AMIs },
  )) {
    imageDetails.push(...(page.Images || []));
  }

  const choices = imageDetails.map((image, index) => ({
    name: `${image.ImageId} - ${image.Description}`,
```

```
    value: index,
  }));

  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an image.",
    choices,
  });

  return imageDetails[selectedIndex];
};

/**
 * @param {import('@aws-sdk/client-ec2').Image} imageDetails
 */
const getCompatibleInstanceTypes = async (imageDetails) => {
  const paginator = paginateDescribeInstanceTypes(
    { client: ec2Client, pageSize: 25 },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          Values: [imageDetails.Architecture],
        },
        { Name: "instance-type", Values: ["*.micro", "*.small"] },
      ],
    },
  );

  const instanceTypes = [];

  for await (const page of paginator) {
    if (page.InstanceTypes.length) {
      instanceTypes.push(...(page.InstanceTypes || []));
    }
  }

  const choices = instanceTypes.map((type, index) => ({
    name: `${type.InstanceType} - Memory:${type.MemoryInfo.SizeInMiB}`,
    value: index,
  }));
```

```
  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an instance type.",
    choices,
  });
  return instanceTypes[selectedIndex];
};

const runInstance = async ({
  keyPairName,
  securityGroupId,
  imageId,
  instanceType,
}) => {
  const command = new RunInstancesCommand({
    KeyName: keyPairName,
    SecurityGroupIds: [securityGroupId],
    ImageId: imageId,
    InstanceType: instanceType,
    MinCount: 1,
    MaxCount: 1,
  });

  const { Instances } = await ec2Client.send(command);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [Instances[0].InstanceId] },
  );
  return Instances[0].InstanceId;
};

const describeInstance = async (instanceId) => {
  const command = new DescribeInstancesCommand({
    InstanceIds: [instanceId],
  });

  const { Reservations } = await ec2Client.send(command);
  return Reservations[0].Instances[0];
};

const displaySSHConnectionInfo = ({ publicIp, keyPairName }) => {
  return `ssh -i ${tmpDirectory}/${keyPairName}.pem ec2-user@${publicIp}`;
```

```
};

const stopInstance = async (instanceId) => {
  const command = new StopInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(command);
  await waitUntilInstanceStopped(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
};

const startInstance = async (instanceId) => {
  const startCommand = new StartInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(startCommand);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
  return await describeInstance(instanceId);
};

const associateAddress = async ({ allocationId, instanceId }) => {
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  const { AssociationId } = await ec2Client.send(command);
  return AssociationId;
};

const disassociateAddress = async (associationId) => {
  const command = new DisassociateAddressCommand({
    AssociationId: associationId,
  });
  try {
    await ec2Client.send(command);
  } catch (err) {
    console.warn(
      `Failed to disassociated address with association id: ${associationId}`,
      err,
    );
  }
};
```

```
const releaseAddress = async (allocationId) => {
  const command = new ReleaseAddressCommand({
    AllocationId: allocationId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Address with allocation ID ${allocationId} released.\n`);
  } catch (err) {
    console.log(
      `Failed to release address with allocation id: ${allocationId}.`,
      err,
    );
  }
};

const restartInstance = async (instanceId) => {
  console.log("Stopping instance.");
  await stopInstance(instanceId);
  console.log("Instance stopped.");
  console.log("Starting instance.");
  const { PublicIpAddress } = await startInstance(instanceId);
  return PublicIpAddress;
};

const terminateInstance = async (instanceId) => {
  const command = new TerminateInstancesCommand({
    InstanceIds: [instanceId],
  });

  try {
    await ec2Client.send(command);
    await waitUntilInstanceTerminated(
      { client: ec2Client },
      { InstanceIds: [instanceId] },
    );
    console.log(`Instance with ID ${instanceId} terminated.\n`);
  } catch (err) {
    console.warn(`Failed to terminate instance ${instanceId}.`, err);
  }
};

const deleteSecurityGroup = async (securityGroupId) => {
```

```
  const command = new DeleteSecurityGroupCommand({
    GroupId: securityGroupId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Security group ${securityGroupId} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete security group ${securityGroupId}.`, err);
  }
};

const deleteKeyPair = async (keyPairName) => {
  const command = new DeleteKeyPairCommand({
    KeyName: keyPairName,
  });

  try {
    await ec2Client.send(command);
    console.log(`Key pair ${keyPairName} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete key pair ${keyPairName}.`, err);
  }
};

const deleteTemporaryDirectory = () => {
  try {
    rmSync(tmpDirectory, { recursive: true });
    console.log(`Temporary directory ${tmpDirectory} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete temporary directory ${tmpDirectory}.`, err);
  }
};

export const main = async () => {
  const keyPairName = "ec2-scenario-key-pair";
  const securityGroupName = "ec2-scenario-security-group";

  let securityGroupId, ipAllocationId, publicIp, instanceId, associationId;

  console.log(wrapText("Welcome to the Amazon EC2 basic usage scenario."));

  try {
    // Prerequisites
```

```
    console.log(
      "Before you launch an instance, you'll need a few things:",
      "\n - A Key Pair",
      "\n - A Security Group",
      "\n - An IP Address",
      "\n - An AMI",
      "\n - A compatible instance type",
      "\n\n I'll go ahead and take care of the first three, but I'll need your help
for the rest.",
    );

    await prompter.confirm({ message: confirmMessage });

    await createKeyPair(keyPairName);
    securityGroupId = await createSecurityGroup(securityGroupName);
    const { PublicIp, AllocationId } = await allocateIpAddress();
    ipAllocationId = AllocationId;
    publicIp = PublicIp;
    const ipAddress = await authorizeSecurityGroupIngress(securityGroupId);

    const { KeyName } = await describeKeyPair(keyPairName);
    const { GroupName } = await describeSecurityGroup(securityGroupName);
    console.log(`# created the key pair ${KeyName}.\n`);
    console.log(
      `# created the security group ${GroupName}`,
      `and allowed SSH access from ${ipAddress} (your IP).\n`,
    );
    console.log(`# allocated ${publicIp} to be used for your EC2 instance.\n`);

    await prompter.confirm({ message: confirmMessage });

    // Creating the instance
    console.log(wrapText("Create the instance."));
    console.log(
      "You get to choose which image you want. Select an amazon-linux-2 image from
the following:",
    );
    const imageDetails = await getAmznLinux2AMIs();
    const instanceTypeDetails = await getCompatibleInstanceTypes(imageDetails);
    console.log("Creating your instance. This can take a few seconds.");
    instanceId = await runInstance({
      keyPairName,
      securityGroupId,
      imageId: imageDetails.ImageId,
```

```
      instanceType: instanceTypeDetails.InstanceType,
    });
    const instanceDetails = await describeInstance(instanceId);
    console.log(`# instance ${instanceId}.\n`);
    console.log(instanceDetails);
    console.log(
      `\nYou should now be able to SSH into your instance from another terminal:`,
      `\n${displaySSHConnectionInfo({
        publicIp: instanceDetails.PublicIpAddress,
        keyPairName,
      })}`,
    );

    await prompter.confirm({ message: confirmMessage });

    // Understanding the IP address.
    console.log(wrapText("Understanding the IP address."));
    console.log(
      "When you stop and start an instance, the IP address will change. I'll restart
your",
      "instance for you. Notice how the IP address changes.",
    );
    const ipAddressAfterRestart = await restartInstance(instanceId);
    console.log(
      `\n Instance started. The IP address changed from
${instanceDetails.PublicIpAddress} to ${ipAddressAfterRestart}`,
      `\n${displaySSHConnectionInfo({
        publicIp: ipAddressAfterRestart,
        keyPairName,
      })}`,
    );
    await prompter.confirm({ message: confirmMessage });
    console.log(
      `If you want to the IP address to be static, you can associate an allocated`,
      `IP address to your instance. I allocated ${publicIp} for you earlier, and now
I'll associate it to your instance.`,
    );
    associationId = await associateAddress({
      allocationId: ipAllocationId,
      instanceId,
    });
    console.log(
      "Done. Now you should be able to SSH using the new IP.\n",
      `${displaySSHConnectionInfo({ publicIp, keyPairName })}`,
```

```
    );
    await prompter.confirm({ message: confirmMessage });
    console.log(
      "I'll restart the server again so you can see the IP address remains the
 same.",
    );
    const ipAddressAfterAssociated = await restartInstance(instanceId);
    console.log(
      `Done. Here's your SSH info. Notice the IP address hasn't changed.`,
      `\n${displaySSHConnectionInfo({
        publicIp: ipAddressAfterAssociated,
        keyPairName,
      })}`,
    );
    await prompter.confirm({ message: confirmMessage });
  } catch (err) {
    console.error(err);
  } finally {
    // Clean up.
    console.log(wrapText("Clean up."));
    console.log("Now I'll clean up all of the stuff I created.");
    await prompter.confirm({ message: confirmMessage });
    console.log("Cleaning up. Some of these steps can take a bit of time.");
    await disassociateAddress(associationId);
    await terminateInstance(instanceId);
    await releaseAddress(ipAllocationId);
    await deleteSecurityGroup(securityGroupId);
    deleteTemporaryDirectory();
    await deleteKeyPair(keyPairName);
    console.log(
      "Done cleaning up. Thanks for staying until the end!",
      "If you have any feedback please use the feedback button in the docs",
      "or create an issue on GitHub.",
    );
  }
};
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - AllocateAddress
  - AssociateAddress

- AuthorizeSecurityGroupIngress
- CreateKeyPair
- CreateSecurityGroup
- DeleteKeyPair
- DeleteSecurityGroup
- DescribeImages
- DescribeInstanceTypes
- DescribeInstances
- DescribeKeyPairs
- DescribeSecurityGroups
- DisassociateAddress
- ReleaseAddress
- RunInstances
- StartInstances
- StopInstances
- TerminateInstances
- UnmonitorInstances

Exemples d'Elastic Load Balancing 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 for JavaScript (v3) avec Elastic Load Balancing.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour Elastic Load Balancing

Les exemples de code suivants montrent comment démarrer avec 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.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeLoadBalancersà la section Référence des AWS SDK for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Création d'un écouteur pour un équilibreur de charge

L'exemple de code suivant montre comment créer un écouteur qui transmet les demandes d'un équilibreur de charge ELB à un groupe cible.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 CreateListenerà la section Référence des AWS SDK for JavaScript API.

Créer un groupe cible

L'exemple de code suivant montre comment créer un groupe cible ELB.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Création d'un Application Load Balancer

L'exemple de code suivant montre comment créer un ELB Application Load Balancer.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer un équilibreur de charge

L'exemple de code suivant montre comment supprimer un équilibreur de charge ELB.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer un groupe cible

L'exemple de code suivant montre comment supprimer un groupe cible ELB.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Décrire les groupes cibles

L'exemple de code suivant montre comment décrire des groupes cibles spécifiques.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenez le point de terminaison d'un équilibreur de charge

L'exemple de code suivant montre comment obtenir le point de terminaison d'un équilibreur de charge ELB.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeLoadBalancersà la section Référence des AWS SDK for JavaScript API.

Obtenez l'état de santé d'un groupe cible

L'exemple de code suivant montre comment obtenir l'état de santé des instances d'un groupe cible ELB.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for 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 instance EC2 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 "url";

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

Créez des étapes pour déployer toutes les ressources.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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 { 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 () => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  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,
      },
    }),
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
}),
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(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
```

```
  new ScenarioAction("getSubnets", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    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"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    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,
```

```
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  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] },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
}),
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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    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 },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
```

```
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
```

```
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
```

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

Créez des étapes pour exécuter la démo.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
```

```
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  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: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
```

```
      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];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
```

```
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      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",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Créez des étapes pour déployer toutes les ressources.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";
```

```
import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} 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 { 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 = [
```

```
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
```

```
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }
}),
new ScenarioAction("detachPolicyFromRole", async (state) => {
  try {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

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

  if (!policy) {
    state.deletePolicyError = new Error(
```

```
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
```

```
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
```

```
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
```

```
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      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.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    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;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
```

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

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

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

```
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

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

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

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

```
}

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

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers

- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Ajout d'une cible

L'exemple de code suivant montre comment ajouter une cible à un EventBridge événement Amazon.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myEventBridgeTarget",
    },
  ],
};

ebevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Success", data);
  }
});
```

- Pour plus de détails sur l'API, reportez-vous PutTargetsà la section Référence des AWS SDK for JavaScript API.

Créer une règle

L'exemple de code suivant montre comment créer une EventBridge règle Amazon.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};
```

```
ebevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- Pour plus de détails sur l'API, reportez-vous PutRuleà la section Référence des AWS SDK for JavaScript API.

Envoyez des événements

L'exemple de code suivant montre comment envoyer EventBridge des événements Amazon.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

ebevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- Pour plus de détails sur l'API, reportez-vous PutEventsà la section Référence des AWS SDK for 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 for JavaScript (v3) avec AWS Glue.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour AWS Glue

Les exemples de code suivants montrent 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 for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Créer un crawler

L'exemple de code suivant montre comment créer un AWS Glue robot d'exploration.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Créer une définition de tâche

L'exemple de code suivant montre comment créer une définition de AWS Glue tâche.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer un crawler

L'exemple de code suivant montre comment supprimer un AWS Glue robot d'exploration.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer une base de données du catalogue de données

L'exemple de code suivant montre comment supprimer une base de données du AWS Glue Data Catalog.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer une définition de tâche

L'exemple de code suivant montre comment supprimer une définition de AWS Glue tâche et toutes les exécutions associées.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer une table d'une base de données

L'exemple de code suivant montre comment supprimer une table d'une AWS Glue Data Catalog base de données.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir un crawler

L'exemple de code suivant montre comment obtenir un AWS Glue robot d'exploration.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir une base de données à partir du catalogue de données

L'exemple de code suivant montre comment obtenir une base de données à partir du AWS Glue Data Catalog.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir une exécution de tâche

L'exemple de code suivant montre comment exécuter une AWS Glue tâche.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir des bases de données à partir du catalogue de données

L'exemple de code suivant montre comment obtenir une liste de bases de données à partir d' AWS Glue Data Catalog.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir une tâche à partir du catalogue de données

L'exemple de code suivant montre comment obtenir une tâche à partir d' AWS Glue Data Catalog.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir des exécutions de tâche

L'exemple de code suivant montre comment exécuter une AWS Glue tâche.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir des tables à partir d'une base de données

L'exemple de code suivant montre comment obtenir des tables à partir d'une base de données dans le AWS Glue Data Catalog.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Répertorier des définitions de tâche

L'exemple de code suivant montre comment répertorier les définitions de AWS Glue tâches.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Démarrer un crawler

L'exemple de code suivant montre comment démarrer un AWS Glue robot d'exploration.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Démarrer une exécution de tâche

L'exemple de code suivant montre comment démarrer l'exécution d'une AWS Glue tâche.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Scénarios

Premiers pas avec les Crawlers et les tâches

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;
  }
};

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 };
  };

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":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "RUNNING":
      break;
    case "SUCCEEDED":
      return;
    default:
      throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
  }

  log(
    `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
  );
  await wait(waitTimeInSeconds);
  return waitForJobRun(getJobRun, jobName, jobRunId);
```

```
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
 view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);

    return { ...context };
  };
```

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);
};

const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (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);
};

const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  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" });
  }
};

const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error)
    )
  );

const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  async (context) => {
```

```
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null })
    );

    if (TableList && TableList.length > 0) {
      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 };
  };

const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error))
  );

const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      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 d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases

- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour HealthImaging

Les exemples de code suivants montrent 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 for 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.

Rubriques

- Actions
- Scénarios

Actions

Ajouter un tag à une ressource

L'exemple de code suivant montre comment ajouter une balise à une HealthImaging ressource.

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 for 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.

Copier un ensemble d'images

L'exemple de code suivant montre comment copier un ensemble HealthImaging d'images.

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.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
```

```
  //      },
  //       datastoreId: 'xxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING',
  //             latestVersionId: '1',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //       },
  //       sourceImageSetProperties: {
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxx/imageset/xxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
  return response;
};
```

Copiez un ensemble d'images sans destination.

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

Copiez un ensemble d'images avec une destination.

```
  try {
```

```
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "4",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

- Pour plus de détails sur l'API, reportez-vous CopyImageSetà la section Référence des AWS SDK for 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.

Création d'un magasin de données

L'exemple de code suivant montre comment créer un magasin HealthImaging de données.

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 for 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.

Supprimer un magasin de données

L'exemple de code suivant montre comment supprimer un magasin HealthImaging de données.

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 for 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.

Supprimer un ensemble d'images

L'exemple de code suivant montre comment supprimer un ensemble HealthImaging d'images.

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: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous DeleteImageSetà la section Référence des AWS SDK for 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.

Obtenir un cadre d'image

L'exemple de code suivant montre comment obtenir un cadre d'image.

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 for 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.

Obtenir les propriétés du magasin de données

L'exemple de code suivant montre comment obtenir les propriétés du magasin de HealthImaging données.

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 for 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.

Obtenir les propriétés du jeu d'images

L'exemple de code suivant montre comment obtenir les propriétés HealthImaging d'un ensemble d'images.

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 = "xxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
```

```
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- Pour plus de détails sur l'API, reportez-vous GetImageSetà la section Référence des AWS SDK for 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.

Obtenir les propriétés des tâches d'importation

L'exemple de code suivant montre comment obtenir les propriétés de la tâche d'importation.

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, consultez GetDICOM ImportJob dans la section AWS SDK for 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.

Obtenir les métadonnées d'un ensemble d'images

L'exemple de code suivant montre comment obtenir des métadonnées pour un ensemble HealthImaging d'images.

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 "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 des ensembles d'images sans version.

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

Obtenez les métadonnées des ensembles d'images avec la 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 for 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.

Importer des données en masse dans un magasin de données

L'exemple de code suivant montre comment importer des données en masse dans un magasin de HealthImaging données.

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, voir StartDICOM ImportJob dans la référence des AWS SDK for 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.

Lister les magasins de données

L'exemple de code suivant montre comment répertorier les magasins de HealthImaging données.

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 for 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.

Lister les versions des ensembles d'images

L'exemple de code suivant montre comment répertorier les versions HealthImaging d'ensembles d'images.

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
  );
```

```
  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- Pour plus de détails sur l'API, reportez-vous ListImageSetVersionsà la section Référence des AWS SDK for 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.

Lister les tâches d'importation pour un magasin de données

L'exemple de code suivant montre comment répertorier les tâches d'importation pour un magasin de HealthImaging données.

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);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
```

```
  // }
  // ]}

  return jobSummaries;
};
```

- Pour plus de détails sur l'API, voir ListDicom ImportJobs dans la référence des AWS SDK for 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.

Répertorie les balises d'une ressource.

L'exemple de code suivant montre comment répertorier les balises d'une HealthImaging ressource.

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 for 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.

Supprimer un tag d'une ressource

L'exemple de code suivant montre comment supprimer une balise d'une HealthImaging ressource.

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 for 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.

Rechercher des ensembles d'images

L'exemple de code suivant montre comment effectuer une recherche dans des ensembles HealthImaging d'images.

SDK pour JavaScript (v3)

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.
 */
export const searchImageSets = async (
```

```
  datastoreId = "xxxxxxxx",
  filters = []
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {
    datastoreId: datastoreId,
    searchCriteria: {
      filters,
    },
  };

  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 #1 : opérateur EQUAL.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [{ DICOMPatientId: "9227465" }],
      operator: "EQUAL",
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}
```

Cas d'utilisation #2 : opérateur BETWEEN utilisant DICOM StudyDate et StudyTime DICOM.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "19900101",
            DICOMStudyTime: "000000",
          },
        },
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "20230901",
            DICOMStudyTime: "000000",
          },
        },
      ],
      operator: "BETWEEN",
```

```
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}
```

Cas d'utilisation #3 : opérateur BETWEEN utilisant CreatedAt. Les études temporelles étaient auparavant poursuivies.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [
        { createdAt: new Date("1985-04-12T23:20:50.52Z") },
        { createdAt: new Date("2023-09-12T23:20:50.52Z") },
      ],
      operator: "BETWEEN",
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}
```

- Pour plus de détails sur l'API, reportez-vous SearchImageSetsà la section Référence des AWS SDK for 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.

Mettre à jour les métadonnées du jeu d'images

L'exemple de code suivant montre comment mettre à jour les métadonnées HealthImaging d'un ensemble d'images.

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.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}') => {
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
            imageSetId: imageSetId,
            latestVersionId: latestVersionId,
            updateImageSetMetadataUpdates: updateMetadata
        })
    );
    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;
};
```

Codez les métadonnées.

```
    const updatableAttributes =
JSON.stringify({
  "SchemaVersion": 1.1,
  "Patient": {
    "DICOM": {
      "PatientName": "Garcia^Gloria"
    }
  }
})

    const updateMetadata = {
        "DICOMUpdates": {
            "updatableAttributes":
                new TextEncoder().encode(updatableAttributes)
        }
    };

    await updateImageSetMetadata("12345678901234567890123456789012",
 "12345678901234567890123456789012",
        "1", updateMetadata);
```

- Pour plus de détails sur l'API, reportez-vous UpdateImageSetMetadataà la section Référence des AWS SDK for 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

Marquage d'un magasin de données

L'exemple de code suivant montre comment étiqueter un magasin de HealthImaging données.

SDK pour JavaScript (v3)

Pour étiqueter 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 le balisage 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 détaguer 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 d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for 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.

Marquer 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 annuler le balisage 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 détaguer 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 d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for 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 for JavaScript (v3) avec IAM.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour IAM

Les exemples de code suivants montrent comment démarrer avec 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) {
      page.Policies.forEach((p) => {
        console.log(`${p.PolicyName}`);
        policyCount++;
      });
    }
  }
```

```
  console.log(`Found ${policyCount} policies.`);
};
```

- Pour plus de détails sur l'API, reportez-vous ListPoliciesà la section Référence des AWS SDK for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Attacher une politique à un rôle

L'exemple de code suivant montre comment associer une politique IAM à un rôle.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous AttachRolePolicyà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        console.log(
          "AmazonDynamoDBFullAccess is already attached to this role."
        );
        process.exit();
      }
```

```
    });
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.attachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous AttachRolePolicyà la section Référence des AWS SDK for JavaScript API.

Attacher une politique en ligne à un rôle

Les exemples de code suivants montrent comment attacher une stratégie en ligne à un rôle 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 { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
```

```
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::some-test-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

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

- Pour plus de détails sur l'API, reportez-vous PutRolePolicyà la section Référence des AWS SDK for JavaScript API.

Créer un fournisseur SAML

L'exemple de code suivant montre comment créer un fournisseur SAML AWS Identity and Access Management (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 { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import * as path from "path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";

const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.
 * For more information on generating this document,
 see https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_providers_create_saml.html#samlstep1.
 */
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
```

```
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });

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

- Pour les détails de l'API, consultez CreateSAMLProvider dans la Référence de l'API AWS SDK for JavaScript .

Créez un groupe

L'exemple de code suivant montre comment créer un groupe 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 { 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 for JavaScript API.

Créer une politique

L'exemple de code suivant montre comment créer une politique 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreatePolicyà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
```

```
      ],
      Resource: "RESOURCE_ARN",
    },
  ],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

iam.createPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreatePolicyà la section Référence des AWS SDK for JavaScript API.

Créer un rôle

L'exemple de code suivant montre comment créer un rôle 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.

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 for JavaScript API.

Créer un rôle lié à un service

L'exemple de code suivant montre comment créer un rôle lié à un service 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.

Créez un rôle lié à un service.

```
import { CreateServiceLinkedRoleCommand, 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,
  });

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

- Pour plus de détails sur l'API, reportez-vous CreateServiceLinkedRoleà la section Référence des AWS SDK for JavaScript API.

Créez un utilisateur

L'exemple de code suivant montre comment créer un utilisateur IAM.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateUserà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateUserà la section Référence des AWS SDK for JavaScript API.

Créer une clé d'accès

L'exemple de code suivant montre comment créer une clé d'accès IAM.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateAccessKeyà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateAccessKeyà la section Référence des AWS SDK for JavaScript API.

Créer un alias pour un compte

L'exemple de code suivant montre comment créer un alias pour un compte 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateAccountAliasà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateAccountAliasà la section Référence des AWS SDK for JavaScript API.

Création d'un profil d'instance

L'exemple de code suivant montre comment créer un profil d'instance 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.

```
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 for JavaScript API.

Supprimer un fournisseur SAML

L'exemple de code suivant montre comment supprimer un fournisseur SAML AWS Identity and Access Management (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 { 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 les détails de l'API, consultez DeleteSAMLProvider dans la Référence de l'API AWS SDK for JavaScript .

Supprimer un groupe

L'exemple de code suivant montre comment supprimer un groupe 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 { 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 for JavaScript API.

Supprimer une politique

L'exemple de code suivant montre comment supprimer une politique 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.

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 for JavaScript API.

Supprimer un rôle

L'exemple de code suivant montre comment supprimer un rôle 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.

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 for JavaScript API.

Supprimer une politique de rôle

L'exemple de code suivant montre comment supprimer une politique de rôle 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 { 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 for JavaScript API.

Supprimer un certificat de serveur

L'exemple de code suivant montre comment supprimer un certificat de serveur 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteServerCertificateà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
```

```
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteServerCertificateà la section Référence des AWS SDK for JavaScript API.

Supprimer un rôle lié à un service

L'exemple de code suivant montre comment supprimer un rôle lié à un service 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 { 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 for JavaScript API.

Suppression d’un utilisateur

L'exemple de code suivant montre comment supprimer un utilisateur IAM.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteUserà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteUserà la section Référence des AWS SDK for JavaScript API.

Supprimer une clé d'accès

L'exemple de code suivant montre comment supprimer une clé d'accès IAM.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteAccessKeyà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

iam.deleteAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteAccessKeyà la section Référence des AWS SDK for JavaScript API.

Supprimer un alias de compte

L'exemple de code suivant montre comment supprimer un alias de compte 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteAccountAliasà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteAccountAliasà la section Référence des AWS SDK for JavaScript API.

Suppression d'un profil d'instance

L'exemple de code suivant montre comment supprimer un profil d'instance 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.

```
    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 for JavaScript API.

Détacher une politique d'un rôle

L'exemple de code suivant montre comment détacher une politique IAM d'un rôle.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DetachRolePolicyà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        var params = {
          PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
          RoleName: process.argv[2],
        };
        iam.detachRolePolicy(params, function (err, data) {
          if (err) {
            console.log("Unable to detach policy from role", err);
          } else {
            console.log("Policy detached from role successfully");
            process.exit();
          }
        });
      }
    });
  }
```

```
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DetachRolePolicyà la section Référence des AWS SDK for JavaScript API.

Obtenir une politique

L'exemple de code suivant montre comment obtenir une politique 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.

- Pour plus de détails sur l'API, reportez-vous GetPolicyà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetPolicyà la section Référence des AWS SDK for JavaScript API.

Obtenir un rôle

L'exemple de code suivant montre comment obtenir un rôle 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.

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 for JavaScript API.

Obtenir un certificat de serveur

L'exemple de code suivant montre comment obtenir un certificat de serveur 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetServerCertificateà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });
```

```
iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetServerCertificateà la section Référence des AWS SDK for JavaScript API.

Obtenir un statut de suppression du rôle lié à un service

L'exemple de code suivant montre comment obtenir le statut de suppression d'un rôle lié à un service AWS Identity and Access Management (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 {
  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 for JavaScript API.

Obtenir des données sur la dernière utilisation d'une clé d'accès

L'exemple de code suivant montre comment obtenir des données relatives à la dernière utilisation d'une clé d'accès IAM.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetAccessKeyLastUsedà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getAccessKeyLastUsed(
```

```
  { AccessKeyId: "ACCESS_KEY_ID" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
    }
  }
);
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetAccessKeyLastUsedà la section Référence des AWS SDK for JavaScript API.

Obtenir la politique de mot de passe du compte

L'exemple de code suivant montre comment obtenir la politique de mot de passe du compte 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.

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 for JavaScript API.

Répertorier les fournisseurs SAML

L'exemple de code suivant montre comment répertorier les fournisseurs SAML pour 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.

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 les détails de l'API, consultez ListSAMLProviders dans la Référence de l'API AWS SDK for JavaScript .

Répertorier la liste des clés d'accès d'un utilisateur

L'exemple de code suivant montre comment répertorier les clés d'accès IAM d'un utilisateur.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListAccessKeysà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};
```

```
iam.listAccessKeys(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListAccessKeysà la section Référence des AWS SDK for JavaScript API.

Répertorier des alias de compte

L'exemple de code suivant montre comment répertorier les alias de comptes 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListAccountAliasesà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
```

```
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListAccountAliasesà la section Référence des AWS SDK for JavaScript API.

Répertorier des groupes

L'exemple de code suivant montre comment répertorier les groupes 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.

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 for JavaScript API.

Répertorier les politiques en ligne pour un rôle

L'exemple de code suivant montre comment répertorier les politiques intégrées pour un rôle 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.

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 for JavaScript API.

Répertorier des politiques

L'exemple de code suivant montre comment répertorier les politiques 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.

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 for JavaScript API.

Répertorier les politiques attachées à un rôle

L'exemple de code suivant montre comment répertorier les politiques associées à un rôle 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.

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 for JavaScript API.

Lister des rôles

L'exemple de code suivant montre comment répertorier les rôles 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.

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 for JavaScript API.

Lister les certificats de serveur

L'exemple de code suivant montre comment répertorier les certificats de serveur 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListServerCertificatesà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListServerCertificatesà la section Référence des AWS SDK for JavaScript API.

Répertorier des utilisateurs

L'exemple de code suivant montre comment répertorier les utilisateurs IAM.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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);
  response.Users?.forEach(({ UserName, CreateDate }) => {
    console.log(`${UserName} created on: ${CreateDate}`);
  });
  return response;
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListUsersà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
      console.log("User " + user.UserName + " created", user.CreateDate);
    });
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListUsersà la section Référence des AWS SDK for JavaScript API.

Mettre à jour un certificat de serveur

L'exemple de code suivant montre comment mettre à jour un certificat de serveur 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateServerCertificateà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

iam.updateServerCertificate(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateServerCertificateà la section Référence des AWS SDK for JavaScript API.

Mettre à jour un utilisateur

L'exemple de code suivant montre comment mettre à jour un utilisateur IAM.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateUserà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

iam.updateUser(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateUserà la section Référence des AWS SDK for JavaScript API.

Mettre à jour une clé d'accès

L'exemple de code suivant montre comment mettre à jour une clé d'accès IAM.

> **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.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateAccessKeyà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};

iam.updateAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous UpdateAccessKeyà la section Référence des AWS SDK for JavaScript API.

Charger un certificat de serveur

L'exemple de code suivant montre comment télécharger un certificat de serveur AWS Identity and Access Management (IAM).

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 "fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
 the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }

    throw err;
  }
};
```

```
/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

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

- Pour plus de détails sur l'API, reportez-vous UploadServerCertificateà la section Référence des AWS SDK for 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 instance EC2 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 sur 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 "url";

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

Créez des étapes pour déployer toutes les ressources.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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 { 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 () => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  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,
        },
      }),
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    );
  }),
  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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
  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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
  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"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    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,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    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] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    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 },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    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) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
```

```
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
```

```
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
```

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

Créez des étapes pour exécuter la démo.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  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: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      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];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      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",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Créez des étapes pour déployer toutes les ressources.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
```

```
} 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 { 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 = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
```

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

```
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

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

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

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

```
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
```

```
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      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.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
```

```
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    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;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
```

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

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

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

```
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

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

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

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

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
```

```
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs

- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Créer un utilisateur et assumer d'un rôle

L'exemple de code suivant montre comment créer un utilisateur et assumer 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 sur 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,
  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";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

export const main = async () => {
  // Create a user. The user has no permissions by default.
  const { User } = await iamClient.send(
    new CreateUserCommand({ UserName: userName }),
  );

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
 STS).
  // It's not best practice to use access keys. For more information, see https://
aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
```

```
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;

  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
      return await listBuckets(s3Client);
    } catch (err) {
      if (err instanceof Error && err.name === "InvalidAccessKeyId") {
        throw err;
      }
    }
  });

  // Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
  // is thrown while the user and access keys are still stabilizing.
  const { Role } = await retry(
    {
      intervalInMs: 2000,
      maxRetries: 60,
    },
    () =>
      iamClient.send(
        new CreateRoleCommand({
          AssumeRolePolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
              {
```

```
              Effect: "Allow",
              Principal: {
                // Allow the previously created user to assume this role.
                AWS: User.Arn,
              },
              Action: "sts:AssumeRole",
            },
          ],
        }),
        RoleName: roleName,
      }),
    ),
);

if (!Role) {
  throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["s3:ListAllMyBuckets"],
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  }),
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
```

```
  }),
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
  listBuckets(s3Client),
);
```

```
  // Clean up.
  await iamClient.send(
    new DetachRolePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeletePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
    }),
  );

  await iamClient.send(
    new DeleteRoleCommand({
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
```

```
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

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 for JavaScript (v3) avec Lambda.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Hello Lambda

Les exemples de code suivants montrent comment démarrer avec Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 for JavaScript API.

Rubriques

- Actions

- Scénarios

Actions

Créer une fonction

L'exemple de code suivant montre comment créer une fonction Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 for JavaScript API.

Supprimer une fonction

L'exemple de code suivant montre comment supprimer une fonction Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 for JavaScript API.

Obtenir une fonction

L'exemple de code suivant montre comment obtenir une fonction Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir GetFunctionla section Référence des AWS SDK for JavaScript API.

Invoquer une fonction

L'exemple de code suivant montre comment invoquer une fonction Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 for JavaScript .

Répertorier des fonctions

L'exemple de code suivant montre comment répertorier les fonctions Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir ListFunctionsla section Référence des AWS SDK for JavaScript API.

Mettre à jour le code de la fonction

L'exemple de code suivant montre comment mettre à jour le code de la fonction Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir UpdateFunctionCodela section Référence des AWS SDK for JavaScript API.

Mettre à jour la configuration de la fonction

L'exemple de code suivant montre comment mettre à jour la configuration de la fonction Lambda.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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,
  });
  return client.send(command);
};
```

- Pour plus de détails sur l'API, voir UpdateFunctionConfigurationla section Référence des AWS SDK for JavaScript API.

Scénarios

Mise en route avec des fonctions

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 sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un rôle AWS Identity and Access Management (IAM) qui accorde à Lambda l'autorisation d'écrire dans les journaux.

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

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 for JavaScript .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Création d'une tâche d'interface par lots

L'exemple de code suivant montre comment créer une tâche d'interface par lots Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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: {         /* required */
    s3DataSource: {   /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional integer*/
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchInferenceJobCommand(createBatchInferenceJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, voir CreateBatchInferenceJobla section Référence des AWS SDK for JavaScript API.

Création d'une tâche de segmentation par lots

L'exemple de code suivant montre comment créer une tâche de segment par lots Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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: {         /* required */
    s3DataSource: {   /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchSegmentJobCommand(createBatchSegmentJobParam));
    console.log("Success", response);
```

```
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, voir CreateBatchSegmentJobla section Référence des AWS SDK for JavaScript API.

Créer une campagne

L'exemple de code suivant montre comment créer une campagne Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateCampaignla section Référence des AWS SDK for JavaScript API.

Créer un jeu de données

L'exemple de code suivant montre comment créer un ensemble de données Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateDatasetla section Référence des AWS SDK for JavaScript API.

Création d'une tâche d'exportation de jeux de données

L'exemple de code suivant montre comment créer une tâche d'exportation d'un ensemble de données Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateDatasetExportJobla section Référence des AWS SDK for JavaScript API.

Création d'un groupe de jeux de données

L'exemple de code suivant montre comment créer un groupe de jeux de données Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 jeux 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, voir CreateDatasetGroupla section Référence des AWS SDK for JavaScript API.

Création d'une tâche d'importation de jeux de données

L'exemple de code suivant montre comment créer une tâche d'importation de jeu de données Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateDatasetImportJobla section Référence des AWS SDK for JavaScript API.

Création d'un schéma de domaine

L'exemple de code suivant montre comment créer un schéma de domaine Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 'fs';
```

```
let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema, /* required */
  domain: 'DOMAIN'   /* required for a domain dataset group, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createDomainSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, voir CreateSchemala section Référence des AWS SDK for JavaScript API.

Création d'un filtre

L'exemple de code suivant montre comment créer un filtre Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateFilterla section Référence des AWS SDK for JavaScript API.

Création d'un recommandeur

L'exemple de code suivant montre comment créer un outil de recommandation Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateRecommenderla section Référence des AWS SDK for JavaScript API.

Création d'un schéma

L'exemple de code suivant montre comment créer un schéma Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema /* required */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
```

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

- Pour plus de détails sur l'API, voir CreateSchemala section Référence des AWS SDK for JavaScript API.

Créez une solution

L'exemple de code suivant montre comment créer une solution Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateSolutionla section Référence des AWS SDK for JavaScript API.

Création d'une version de solution

L'exemple de code suivant montre comment créer une solution Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateSolutionVersionla section Référence des AWS SDK for JavaScript API.

Création d'un outil de suivi des événements

L'exemple de code suivant montre comment créer un outil de suivi d'événements Amazon Personalize.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir CreateEventTrackerla section Référence des AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Importer des éléments dans un ensemble de données

L'exemple de code suivant montre comment importer progressivement des articles dans un ensemble de données Amazon Personalize Events.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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.
var putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /*  required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
 "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, voir PutItemsla section Référence des AWS SDK for JavaScript API.

Importer des données d'événements d'interaction en temps réel

L'exemple de code suivant montre comment importer des données d'événements d'interaction en temps réel dans Amazon Personalize Events.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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.
var putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
```

```
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, voir PutEventsla section Référence des AWS SDK for JavaScript API.

Importer un utilisateur de manière incrémentielle

L'exemple de code suivant montre comment importer progressivement un utilisateur dans Amazon Personalize Events Events Events.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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.
var putUsersParam = {
  datasetArn: "DATASET_ARN",
  users: [
```

```
    {
      userId: "USER_ID",
      properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Pour plus de détails sur l'API, voir PutUsersla section Référence des AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Obtenir des recommandations (groupe de jeux de données personnalisé)

L'exemple de code suivant montre comment obtenir des recommandations classées par Amazon Personalize Runtime Runtime.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, voir GetPersonalizedRankingla section Référence des AWS SDK for JavaScript API.

Obtenir des recommandations d'un recommandeur (groupe de jeux de données de domaine)

L'exemple de code suivant montre comment obtenir les recommandations d'Amazon Personalize Runtime Runtime.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 jeux 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 recommandeur 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, voir GetRecommendationsla section Référence des AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Envoyer des e-mails et des SMS

L'exemple de code suivant montre comment envoyer des e-mails et des SMS avec Amazon Pinpoint.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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";
//Set the MediaConvert Service Object
const pinClient = new PinpointClient({ region: REGION });
```

```
export { pinClient };
```

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.
var subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
 using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding for the subject line and message body of the email.
var charset = "UTF-8";

const params = {
  ApplicationId: projectId,
  MessageRequest: {
```

```
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));

    const {
      MessageResponse: { Result },
    } = data;

    const recipientResult = Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    } else {
      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";

("use strict");

/* The phone number or short code to send the message from. The phone number
 or short code that you specify has to be associated with your Amazon Pinpoint
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number.  For best results, you should specify the phone
 number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";
```

```
/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderID";

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    return data; // For unit tests.
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  } catch (err) {
    console.log(err);
  }
};
run();
```

- Pour plus de détails sur l'API, voir SendMessagesla section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyer un e-mail.

```
"use strict";

const AWS = require("aws-sdk");

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
var toAddress = "recipient@example.com";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
----------------------------------------------------
```

```
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https:\/\/aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: appId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: senderAddress,
        SimpleEmail: {
          Subject: {
```

```
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

//Try to send the email.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
  } else {
    console.log(
      "Email sent! Message ID: ",
      data["MessageResponse"]["Result"][toAddress]["MessageId"]
    );
  }
});
```

Envoyer un SMS.

```
"use strict";

var AWS = require("aws-sdk");

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";
```

```
// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
var originationNumber = "+12065550199";

// The recipient's phone number.  For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderID";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();
```

```
// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

//Try to send the message.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
    // Otherwise, show the unique ID for the message.
  } else {
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  }
});
```

- Pour plus de détails sur l'API, voir SendMessagesla section Référence des AWS SDK for JavaScript API.

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Créer un cluster

L'exemple de code suivant montre comment créer un cluster Amazon Redshift.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 { RedshiftClient } = require("@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, voir CreateClusterla section Référence des AWS SDK for JavaScript API.

Supprimer un cluster

L'exemple de code suivant montre comment supprimer un cluster Amazon Redshift.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 { RedshiftClient } = require("@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, voir DeleteClusterla section Référence des AWS SDK for JavaScript API.

Décrivez vos clusters

L'exemple de code suivant montre comment décrire vos clusters Amazon Redshift.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 { RedshiftClient } = require("@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, voir DescribeClustersla section Référence des AWS SDK for JavaScript API.

Modifier un cluster

L'exemple de code suivant montre comment modifier un cluster Amazon Redshift.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 { RedshiftClient } = require("@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, voir ModifyClusterla section Référence des AWS SDK for JavaScript API.

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 for JavaScript (v3) avec Amazon S3.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Hello Amazon S3

Les exemples de code suivants montrent comment démarrer avec Amazon S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

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

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new S3Client({});

export const helloS3 = async () => {
  const command = new ListBucketsCommand({});

  const { Buckets } = await client.send(command);
  console.log("Buckets: ");
  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
  return Buckets;
};
```

- Pour plus de détails sur l'API, voir ListBucketsla section Référence des AWS SDK for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Ajouter des règles CORS à un compartiment

L'exemple de code suivant montre comment ajouter des règles de partage de ressources entre origines (CORS) à un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

// By default, Amazon S3 doesn't allow cross-origin requests. Use this command
// to explicitly allow cross-origin requests.
export const main = async () => {
  const command = new PutBucketCorsCommand({
    Bucket: "test-bucket",
    CORSConfiguration: {
      CORSRules: [
        {
          // Allow all headers to be sent to this bucket.
          AllowedHeaders: ["*"],
          // Allow only GET and PUT methods to be sent to this bucket.
          AllowedMethods: ["GET", "PUT"],
          // Allow only requests from the specified origin.
          AllowedOrigins: ["https://www.example.com"],
          // Allow the entity tag (ETag) header to be returned in the response. The
 ETag header
          // The entity tag represents a specific version of the object. The ETag
 reflects
          // changes only to the contents of an object, not its metadata.
          ExposeHeaders: ["ETag"],
          // How long the requesting browser should cache the preflight response.
 After
          // this time, the preflight request will have to be made again.
          MaxAgeSeconds: 3600,
        },
      ],
    },
  });
```

```
  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir PutBucketCorsla section Référence des AWS SDK for JavaScript API.

Ajouter une politique à un compartiment

L'exemple de code suivant montre comment ajouter une politique à un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutBucketPolicyCommand({
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "AllowGetObject",
          // Allow this particular user to call GetObject on any object in this
 bucket.
          Effect: "Allow",
```

```
          Principal: {
            AWS: "arn:aws:iam::ACCOUNT-ID:user/USERNAME",
          },
          Action: "s3:GetObject",
          Resource: "arn:aws:s3:::BUCKET-NAME/*",
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: "BUCKET-NAME",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir PutBucketPolicyla section Référence des AWS SDK for JavaScript API.

Copier un objet depuis un compartiment vers un autre

L'exemple de code suivant montre comment copier un objet S3 d'un compartiment à un autre.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";
```

```
const client = new S3Client({});

export const main = async () => {
  const command = new CopyObjectCommand({
    CopySource: "SOURCE_BUCKET/SOURCE_OBJECT_KEY",
    Bucket: "DESTINATION_BUCKET",
    Key: "NEW_OBJECT_KEY",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, voir CopyObjectla section Référence des AWS SDK for JavaScript API.

Création d'un compartiment

L'exemple de code suivant montre comment créer un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 { CreateBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CreateBucketCommand({
```

```
    // The name of the bucket. Bucket names are unique and have several other
 constraints.
    // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
bucketnamingrules.html
    Bucket: "bucket-name",
  });

  try {
    const { Location } = await client.send(command);
    console.log(`Bucket created with location ${Location}`);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir CreateBucketla section Référence des AWS SDK for JavaScript API.

Supprimer une stratégie d'un compartiment

L'exemple de code suivant montre comment supprimer une politique d'un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

// This will remove the policy from the bucket.
export const main = async () => {
  const command = new DeleteBucketPolicyCommand({
```

```
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir DeleteBucketPolicyla section Référence des AWS SDK for JavaScript API.

Supprimer un compartiment vide

L'exemple de code suivant montre comment supprimer un compartiment S3 vide.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Delete a bucket.
export const main = async () => {
  const command = new DeleteBucketCommand({
    Bucket: "test-bucket",
  });

  try {
```

```
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir DeleteBucketla section Référence des AWS SDK for JavaScript API.

Supprimer un objet

L'exemple de code suivant montre comment supprimer un objet S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectCommand({
    Bucket: "test-bucket",
    Key: "test-key.txt",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
```

```
  }
};
```

- Pour plus de détails sur l'API, voir DeleteObjectla section Référence des AWS SDK for JavaScript API.

Suppression de plusieurs objets

L'exemple de code suivant montre comment supprimer plusieurs objets d'un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectsCommand({
    Bucket: "test-bucket",
    Delete: {
      Objects: [{ Key: "object1.txt" }, { Key: "object2.txt" }],
    },
  });

  try {
    const { Deleted } = await client.send(command);
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted
 objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (err) {
```

```
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, voir DeleteObjectsla section Référence des AWS SDK for JavaScript API.

Supprimer la configuration du site web d'un compartiment

L'exemple de code suivant montre comment supprimer la configuration du site Web d'un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Disable static website hosting on the bucket.
export const main = async () => {
  const command = new DeleteBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir DeleteBucketWebsitela section Référence des AWS SDK for JavaScript API.

Obtenir des règles CORS pour un compartiment

L'exemple de code suivant montre comment obtenir des règles de partage de ressources entre origines (CORS) pour un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketCorsCommand({
    Bucket: "test-bucket",
  });

  try {
    const { CORSRules } = await client.send(command);
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `\n${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders.join(" ")}`,
        `\nAllowedMethods: ${cr.AllowedMethods.join(" ")}`,
        `\nAllowedOrigins: ${cr.AllowedOrigins.join(" ")}`,
        `\nExposeHeaders: ${cr.ExposeHeaders.join(" ")}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
```

```
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir GetBucketCorsla section Référence des AWS SDK for JavaScript API.

Obtenir un objet depuis un compartiment

L'exemple de code suivant montre comment lire les données d'un objet dans un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
  });

  try {
    const response = await client.send(command);
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (err) {
```

```
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir GetObjectla section Référence des AWS SDK for JavaScript API.

Obtenir l'ACL d'un compartiment

L'exemple de code suivant montre comment obtenir la liste de contrôle d'accès (ACL) d'un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketAclCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir GetBucketAcllla section Référence des AWS SDK for JavaScript API.

Obtenir la politique d'un compartiment

L'exemple de code suivant montre comment obtenir la politique d'un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const { Policy } = await client.send(command);
    console.log(JSON.parse(Policy));
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir GetBucketPolicylla section Référence des AWS SDK for JavaScript API.

Obtenir la configuration du site web pour un compartiment

L'exemple de code suivant montre comment obtenir la configuration du site Web pour un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const { ErrorDocument, IndexDocument } = await client.send(command);
    console.log(
      `Your bucket is set up to host a website. It has an error document:`,
      `${ErrorDocument.Key}, and an index document: ${IndexDocument.Suffix}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, voir GetBucketWebsitela section Référence des AWS SDK for JavaScript API.

Lister les compartiments

L'exemple de code suivant montre comment répertorier les compartiments S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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 { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

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

  try {
    const { Owner, Buckets } = await client.send(command);
    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`,
    );
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir ListBucketsla section Référence des AWS SDK for JavaScript API.

Lister des objets dans un compartiment

L'exemple de code suivant montre comment répertorier des objets dans un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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,
  // This command supersedes the ListObjectsCommand and is the recommended way to
 list objects.
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListObjectsV2Command({
    Bucket: "my-bucket",
    // The default and maximum number of keys returned is 1000. This limits it to
    // one for demonstration purposes.
    MaxKeys: 1,
  });

  try {
    let isTruncated = true;

    console.log("Your bucket contains the following objects:\n");
    let contents = "";

    while (isTruncated) {
      const { Contents, IsTruncated, NextContinuationToken } =
        await client.send(command);
      const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
      contents += contentsList + "\n";
      isTruncated = IsTruncated;
      command.input.ContinuationToken = NextContinuationToken;
    }
```

```
    console.log(contents);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour plus de détails sur l'API, voir ListObjectsV2 dans le manuel de référence des AWS SDK for JavaScript API.

Définir une nouvelle ACL pour un compartiment

L'exemple de code suivant montre comment définir une nouvelle liste de contrôle d'accès (ACL) pour un compartiment S3.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus sur 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,
  GetBucketAclCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const client = new S3Client({});

// Most Amazon S3 use cases don't require the use of access control lists (ACLs).
// We recommend that you disable ACLs, except in unusual circumstances where
// you need to control access for each object individually.
// Consider a policy instead. For more information see https://docs.aws.amazon.com/
AmazonS3/latest/userguide/bucket-policies.html.
export const main = async () => {
  // Grant a user READ access to a bucket.
  const command = new PutBucketAclCommand({
```

```
    Bucket: "test-bucket",
    AccessControlPolicy: {
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
 AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
            ID: "canonical-id-1",
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "FULL_CONTROL",
        },
      ],
      Owner: {
        ID: "canonical-id-2",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir PutBucketAcll a section Référence des AWS SDK for JavaScript API.

Définir la configuration du site web pour un compartiment

L'exemple de code suivant montre comment définir la configuration du site Web pour un compartiment 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.

Définissez la configuration du site web.

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

const client = new S3Client({});

// Set up a bucket as a static website.
// The bucket needs to be publicly accessible.
export const main = async () => {
  const command = new PutBucketWebsiteCommand({
    Bucket: "test-bucket",
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
        // A suffix that is appended to a request that is for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.

- Pour plus de détails sur l'API, voir PutBucketWebsitela section Référence des AWS SDK for JavaScript API.

Charger un objet dans un compartiment

L'exemple de code suivant montre comment télécharger un objet dans un compartiment 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.

Chargez l'objet.

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

const client = new S3Client({});

export const main = async () => {
  const command = new PutObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
    Body: "Hello S3!",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir PutObjectla section Référence des AWS SDK for JavaScript API.

Scénarios

Créer une URL présignée

L'exemple de code suivant montre comment créer une URL présignée pour Amazon S3 et télécharger 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 "https";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
  return formatUrl(signedUrlObject);
};
```

```
const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

function put(url, data) {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          resolve(responseBody);
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
}

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.txt";

  // There are two ways to generate a presigned URL.
  // 1. Use createPresignedUrl without the S3 client.
  // 2. Use getSignedUrl in conjunction with the S3 client and GetObjectCommand.
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });
```

```
    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    // After you get the presigned URL, you can provide your own file
    // data. Refer to put() above.
    console.log("Calling PUT using presigned URL without client");
    await put(noClientUrl, "Hello World");

    console.log("Calling PUT using presigned URL with client");
    await put(clientUrl, "Hello World");

    console.log("\nDone. Check your S3 console.");
  } catch (err) {
    console.error(err);
  }
};
```

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 });
};

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.jpg";

  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    console.log("Presigned URL without client");
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (err) {
    console.error(err);
  }
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.

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,
  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, voir ListObjectsla section Référence des AWS SDK for JavaScript API.

Démarrer avec les compartiments et les objets

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 "url";
import { readdirSync, readFileSync, writeFileSync } from "fs";

// Local helper utils.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

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

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 } from "@inquirer/prompts";

export class Prompter {
  /**
```

```
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  select(options) {
    return select(options);
  }

  /**
   * @param {{ message: string }} options
   */
  input(options) {
    return input(options);
  }

  /**
   * @param {string} prompt
   */
  checkContinue = async (prompt = "") => {
    const prefix = prompt && prompt + " ";
    let ok = await this.confirm({
      message: `${prefix}Continue?`,
    });
    if (!ok) throw new Error("Exiting...");
  };

  /**
   * @param {{ message: string }} options
   */
  confirm(options) {
    return confirm(options);
  }

  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  checkbox(options) {
    return checkbox(options);
  }
}

export const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};
```

Les objets dans S3 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 (let 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 l'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;
  } else {
    const copy = async () => {
      try {
        const sourceBucket = await prompter.input({
          message: "Enter source bucket name:",
        });
        const sourceKey = await prompter.input({
          message: "Enter source key:",
        });
        const destinationKey = await prompter.input({
          message: "Enter destination key:",
        });

        const command = new CopyObjectCommand({
          Bucket: destinationBucket,
          CopySource: `${sourceBucket}/${sourceKey}`,
          Key: destinationKey,
```

```
        });
        await s3Client.send(command);
        await copyFileFromBucket({ destinationBucket });
      } catch (err) {
        console.error(`Copy error.`);
        console.error(err);
        const retryAnswer = await prompter.confirm({ message: "Try again?" });
        if (retryAnswer) {
          await copy();
        }
      }
    };
    await copy();
  }
};
```

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 (let content of Contents) {
    const obj = await s3Client.send(
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
    );
    writeFileSync(
      `${path}/${content.Key}`,
      await obj.Body.transformToByteArray(),
    );
  }
  console.log("Files downloaded successfully.\n");
};
```

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 d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - 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 {
  CreateMultipartUploadCommand,
  UploadPartCommand,
  CompleteMultipartUploadCommand,
  AbortMultipartUploadCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

export const main = async () => {
  const s3Client = new S3Client({});
  const bucketName = "test-bucket";
  const key = "multipart.txt";
  const str = createString();
  const buffer = Buffer.from(str, "utf8");

  let uploadId;

  try {
    const multipartUpload = await s3Client.send(
      new CreateMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );

    uploadId = multipartUpload.UploadId;
```

```
    const uploadPromises = [];
    // Multipart uploads require a minimum size of 5 MB per part.
    const partSize = Math.ceil(buffer.length / 5);

    // Upload each part.
    for (let i = 0; i < 5; i++) {
      const start = i * partSize;
      const end = start + partSize;
      uploadPromises.push(
        s3Client
          .send(
            new UploadPartCommand({
              Bucket: bucketName,
              Key: key,
              UploadId: uploadId,
              Body: buffer.subarray(start, end),
              PartNumber: i + 1,
            }),
          )
          .then((d) => {
            console.log("Part", i + 1, "uploaded");
            return d;
          }),
      );
    }

    const uploadResults = await Promise.all(uploadPromises);

    return await s3Client.send(
      new CompleteMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
        UploadId: uploadId,
        MultipartUpload: {
          Parts: uploadResults.map(({ ETag }, i) => ({
            ETag,
            PartNumber: i + 1,
          })),
        },
      }),
    );

    // Verify the output by downloading the file from the Amazon Simple Storage
  Service (Amazon S3) console.
```

```
    // Because the output is a 25 MB string, text editors might struggle to open the
 file.
  } catch (err) {
    console.error(err);

    if (uploadId) {
      const abortCommand = new AbortMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
        UploadId: uploadId,
      });

      await s3Client.send(abortCommand);
    }
  }
};
```

Téléchargez un fichier volumineux.

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

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;

export const getObjectRange = ({ bucket, key, start, end }) => {
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: parseInt(start),
    end: parseInt(end),
    length: parseInt(length),
```

```
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

// When downloading a large file, you might want to break it down into
// smaller pieces. Amazon S3 accepts a Range header to specify the start
// and end of the byte range to be downloaded.
const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url))
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    console.log(`Downloading bytes ${nextRange.start} to ${nextRange.end}`);

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
    });

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

export const main = async () => {
  await downloadInChunks({
    bucket: "my-cool-bucket",
    key: "my-cool-object.txt",
  });
};
```

Exemples de S3 Glacier 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 for JavaScript (v3) avec S3 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Créer un coffre

L'exemple de code suivant montre comment créer un coffre-fort Amazon S3 Glacier.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir CreateVaultla section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object
```

```
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
// Call Glacier to create the vault
glacier.createVault({ vaultName: "YOUR_VAULT_NAME" }, function (err) {
  if (!err) {
    console.log("Created vault!");
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir CreateVaultla section Référence des AWS SDK for JavaScript API.

Chargement d'une archive sur un coffre

L'exemple de code suivant montre comment télécharger une archive dans un coffre-fort Amazon S3 Glacier.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir UploadArchivela section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object and buffer
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
buffer = Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer

var params = { vaultName: "YOUR_VAULT_NAME", body: buffer };
```

```
// Call Glacier to upload the archive.
glacier.uploadArchive(params, function (err, data) {
  if (err) {
    console.log("Error uploading archive!", err);
  } else {
    console.log("Archive ID", data.archiveId);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, voir UploadArchivela section Référence des AWS SDK for JavaScript API.

SageMaker 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 for JavaScript (v3) avec SageMaker.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour SageMaker

Les exemples de code suivants montrent comment démarrer avec SageMaker.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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, voir ListNotebookInstancesla section Référence des AWS SDK for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Crée un pipeline.

L'exemple de code suivant montre comment créer ou mettre à jour un pipeline dans SageMaker.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 SageMaker pipeline 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,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  const { PipelineArn } = await sagemakerClient.send(
    new CreatePipelineCommand({
      PipelineName: name,
      PipelineDefinition: pipelineDefinition,
      RoleArn: roleArn,
    }),
  );

  return {
    arn: PipelineArn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - CreatePipeline
  - UpdatePipeline

Supprime un pipeline.

L'exemple de code suivant montre comment supprimer un pipeline dans SageMaker.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de suppression d'un SageMaker pipeline. 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, voir DeletePipelinela section Référence des AWS SDK for JavaScript API.

Décrire l'exécution d'un pipeline

L'exemple de code suivant montre comment décrire l'exécution d'un pipeline dans SageMaker.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 SageMaker pipeline 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}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- Pour plus de détails sur l'API, voir DescribePipelineExecutionla section Référence des AWS SDK for JavaScript API.

Exécuter un pipeline

L'exemple de code suivant montre comment démarrer l'exécution d'un pipeline dans SageMaker.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 SageMaker pipeline.

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
```

```
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
```

```
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}
```

- Pour plus de détails sur l'API, voir StartPipelineExecutionla section Référence des AWS SDK for JavaScript API.

Scénarios

Commencez par les travaux et les pipelines géospatiaux

L'exemple de code suivant illustre comment :

- Configurez les ressources d'un pipeline.
- Configurez un pipeline qui exécute une tâche géospatiale.
- Démarrez l'exécution d'un pipeline.
- Surveillez le statut de l'exécution.
- Affichez la sortie du pipeline.
- Nettoyez les ressources.

Pour plus d'informations, voir Création et exécution de SageMaker pipelines à l'aide de kits de AWS développement logiciel (SDK) sur 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 SageMaker client pour gérer un pipeline.

```
import { readFileSync } from "fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";
```

```
import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-
geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry, wait } 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 { Role } = await iamClient.send(
    new CreateRoleCommand({
      RoleName: name,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Action: ["sts:AssumeRole"],
            Principal: { Service: ["lambda.amazonaws.com"] },
          },
        ],
      }),
    }),
  );

  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 policy = {
    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 createPolicyCommand = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify(policy),
    PolicyName: name,
  });

  const { Policy } = await iamClient.send(createPolicyCommand);
  return {
    arn: Policy.Arn,
    policy,
    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`;

  const command = new CreateFunctionCommand({
    Code: {
      ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
    },
    Runtime: Runtime.nodejs18x,
    Handler: "index.handler",
    Layers: [layerVersionArn],
    FunctionName: name,
    Role: roleArn,
  });

  // 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 },
    () => lambdaClient.send(command),
  );

  return {
    arn: FunctionArn,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteFunctionCommand({ FunctionName: name }),
```

```
      );
    },
  };
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
 props
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
  const s3Path = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../../workflows/sagemaker_pipelines/resources/latlongtest.csv`;

  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "input/sample_data.csv",
      Body: readFileSync(s3Path),
    }),
  );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient}} props
 */
export async function createSagemakerRole({ name, iamClient }) {
  const command = 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",
```

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

  const { Role } = await iamClient.send(command);
  // Wait for the role to be ready.
  await wait(10);

  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 policy = {
    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 createPolicyCommand = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify(policy),
    PolicyName: name,
  });

  const { Policy } = await iamClient.send(createPolicyCommand);
  return {
    arn: Policy.Arn,
    policy,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: Policy.Arn }));
    },
  };
}

/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
```

```
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  const { PipelineArn } = await sagemakerClient.send(
    new CreatePipelineCommand({
      PipelineName: name,
      PipelineDefinition: pipelineDefinition,
      RoleArn: roleArn,
    }),
  );

  return {
    arn: PipelineArn,
    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 { QueueUrl } = await sqsClient.send(
    new CreateQueueCommand({
      QueueName: name,
      Attributes: {
        DelaySeconds: "5",
        ReceiveMessageWaitTimeSeconds: "5",
        VisibilityTimeout: "300",
      },
```

```
    }),
  );

  const { Attributes } = await sqsClient.send(
    new GetQueueAttributesCommand({
      QueueUrl,
      AttributeNames: ["QueueArn"],
    }),
  );

  return {
    queueUrl: QueueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{lambdaName: string, queueArn: string, lambdaClient: import('@aws-sdk/
client-lambda').LambdaClient, sqsClient: import('@aws-sdk/client-sqs').SQSClient}}
 props
 */
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
}) {
  const { UUID } = await lambdaClient.send(
    new CreateEventSourceMappingCommand({
      EventSourceArn: queueArn,
      FunctionName: lambdaName,
    }),
  );

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
          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}} props
 */
export async function createS3Bucket({ name, s3Client }) {
  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}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });
```

```
  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-
s3').S3Client}} param0
 */
export async function getObject({ bucket, s3Client }) {
  const prefix = "output/";
  const { Contents } = await s3Client.send(
    new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
  );

  if (!Contents.length) {
    throw new Error("No objects found in bucket.");
```

```
  }

  // Find the CSV file.
  const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

  if (!outputObject) {
    throw new Error(`No CSV file found in bucket with the prefix "${prefix}".`);
  }

  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: outputObject.Key,
    }),
  );

  return Body.transformToString();
}
```

Cette fonction est un extrait d'un fichier qui utilise les fonctions de bibliothèque précédentes pour configurer un SageMaker pipeline, 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 {
      // Run all of the clean up functions. If any fail, we log the error and
 continue.
      // This ensures all clean up functions are run.
      this.logger.logSeparator();
      const doCleanUp = await this.prompter.confirm({
        message: "Clean up resources?",
```

```
      });
      if (doCleanUp) {
        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,
    });
    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,
    });

    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 d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Obtenir une valeur secrète

L'exemple de code suivant montre comment obtenir une valeur de Secrets Secrets 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 {
  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, voir GetSecretValuela section Référence des AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Créer un filtre de réception

L'exemple de code suivant montre comment créer un filtre de réception Amazon SES qui bloque les messages provenant d'une adresse IP ou d'une plage d'adresses IP.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 (err) {
    console.log("Failed to create filter.", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, voir CreateReceiptFilterla section Référence des AWS SDK for JavaScript API.

Création d'une règle de réception

L'exemple de code suivant montre comment créer une règle de réception Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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, voir CreateReceiptRulela section Référence des AWS SDK for JavaScript API.

Créer un jeu de règles de réception

L'exemple de code suivant montre comment créer un jeu de règles de réception Amazon SES afin d'organiser les règles appliquées aux e-mails entrants.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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, voir CreateReceiptRuleSetla section Référence des AWS SDK for JavaScript API.

Créer un modèle d'e-mail

L'exemple de code suivant montre comment créer un modèle d'e-mail Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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, voir CreateTemplatela section Référence des AWS SDK for JavaScript API.

Supprimer un filtre de réception

L'exemple de code suivant montre comment supprimer un filtre de réception Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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, voir DeleteReceiptFilterla section Référence des AWS SDK for JavaScript API.

Supprimer une règle de réception

L'exemple de code suivant montre comment supprimer une règle de réception Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer un jeu de règles

L'exemple de code suivant montre comment supprimer un jeu de règles Amazon SES et toutes les règles qu'il contient.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer un modèle d'e-mail

L'exemple de code suivant montre comment supprimer un modèle d'e-mail Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Supprimer une identité

L'exemple de code suivant montre comment supprimer une identité Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Obtenir un modèle d'e-mail existant

L'exemple de code suivant montre comment obtenir un modèle d'e-mail Amazon SES existant.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 (err) {
    console.log("Failed to get email template.", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous GetTemplateà la section Référence des AWS SDK for JavaScript API.

Répertorier les modèles d'e-mail

L'exemple de code suivant montre comment répertorier des modèles d'e-mail Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Répertorier les identités

L'exemple de code suivant montre comment répertorier les identités Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Répertorier les filtres de réception

L'exemple de code suivant montre comment répertorier des filtres de réception Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Envoyer un e-mail basé sur un modèle en bloc

L'exemple de code suivant montre comment envoyer un e-mail basé sur un modèle à plusieurs destinations avec Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 (err) {
    console.log("Failed to send bulk template email", err);
    return err;
  }
```

```
};
```

- Pour plus de détails sur l'API, reportez-vous SendBulkTemplatedEmailà la section Référence des AWS SDK for JavaScript API.

Send email (Envoyer un e-mail)

L'exemple de code suivant montre comment envoyer un e-mail avec Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 (e) {
    console.error("Failed to send email.");
    return e;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous SendEmailà la section Référence des AWS SDK for JavaScript API.

Envoi d'e-mails bruts

L'exemple de code suivant montre comment envoyer un e-mail brut avec Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Envoyer un e-mail basé sur un modèle

L'exemple de code suivant montre comment envoyer un e-mail basé sur un modèle avec Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 (err) {
    console.log("Failed to send template email", err);
    return err;
  }
};
```

- Pour plus de détails sur l'API, reportez-vous SendTemplatedEmailà la section Référence des AWS SDK for JavaScript API.

Mettre à jour un modèle d'e-mail

L'exemple de code suivant montre comment mettre à jour un modèle d'e-mail Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Vérifier l'identité d'un domaine

L'exemple de code suivant montre comment vérifier une identité de domaine avec Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Vérifier une identité d'e-mail

L'exemple de code suivant montre comment vérifier une identité d'e-mail avec Amazon SES.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Exemples d'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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Hello Amazon SNS

Les exemples de code suivants montrent comment démarrer avec 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 for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Vérifier si un numéro de téléphone est désactivé

L'exemple de code suivant montre comment vérifier si un numéro de téléphone est désactivé pour ne pas recevoir de messages 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.

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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.

- Pour plus de détails sur l'API, reportez-vous CheckIfPhoneNumberIsOptedOutà la section Référence des AWS SDK for JavaScript API.

Confirmer qu'un propriétaire de point de terminaison souhaite recevoir des messages

L'exemple de code suivant montre comment confirmer que le propriétaire d'un point de terminaison souhaite recevoir des messages Amazon SNS en validant le jeton envoyé au point de terminaison par une action d'abonnement antérieure.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous ConfirmSubscriptionà la section Référence des AWS SDK for JavaScript API.

Créer une rubrique

L'exemple de code suivant montre comment créer une rubrique 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.

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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous CreateTopicà la section Référence des AWS SDK for JavaScript API.

Supprimer un abonnement

L'exemple de code suivant montre comment supprimer un abonnement 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.

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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour de plus amples informations sur l'API, consultez Se désabonner dans Référence de l'API AWS SDK for JavaScript .

Supprimer une rubrique

L'exemple de code suivant montre comment supprimer une rubrique Amazon SNS et tous les abonnements à cette rubrique.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous DeleteTopicà la section Référence des AWS SDK for JavaScript API.

Obtenir les propriétés d'une rubrique

L'exemple de code suivant montre comment obtenir les propriétés d'une rubrique 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.

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

- Pour de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous GetTopicAttributesà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();
```

```
// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

- Pour de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous GetTopicAttributesà la section Référence des AWS SDK for JavaScript API.

Obtenir les paramètres d'envoi de messages SMS

L'exemple de code suivant montre comment obtenir les paramètres d'envoi de messages SMS 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.

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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour de plus amples informations sur l'API, consultez GetSMSAttributes dans Référence de l'API AWS SDK for JavaScript .

Lister les abonnés d'une rubrique

L'exemple de code suivant montre comment récupérer la liste des abonnés d'une rubrique 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.

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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous ListSubscriptionsà la section Référence des AWS SDK for JavaScript API.

Liste des rubriques

L'exemple de code suivant montre comment répertorier les rubriques 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.

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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous ListTopicsà la section Référence des AWS SDK for JavaScript API.

Publication d'un message avec un attribut

L'exemple de code suivant montre comment publier un message avec un attribut à l'aide d'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.

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, deduplicationId, choices;

  if (this.isFifo) {
    await this.logger.log(MESSAGES.groupIdNotice);
```

```
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );
```

```
    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
```

- Pour de plus amples informations sur l'API, consultez Publier dans Référence de l'API AWS SDK for JavaScript .

Publier dans une rubrique

L'exemple de code suivant montre comment publier des messages sur une rubrique 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.

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;
};
```

- Pour de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour de plus amples informations sur l'API, consultez Publier dans Référence de l'API AWS SDK for JavaScript .

Définir les paramètres par défaut pour l'envoi de messages SMS

L'exemple de code suivant montre comment définir les paramètres par défaut pour l'envoi de SMS via 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.

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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour les détails d'API, consultez SetSMSAttributes dans Référence de l'API AWS SDK for JavaScript .

Définir les attributs de la rubrique

L'exemple de code suivant montre comment définir les attributs des rubriques 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.

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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour plus de détails sur l'API, reportez-vous SetTopicAttributesà la section Référence des AWS SDK for JavaScript API.

Abonner une fonction Lambda à une rubrique

L'exemple de code suivant montre comment abonner une fonction Lambda afin qu'elle reçoive des notifications d'une rubrique 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.

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 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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour de plus amples informations sur l'API, consultez S'abonner dans AWS SDK for JavaScript Référence de l'API.

Abonner une application mobile à une rubrique

L'exemple de code suivant montre comment abonner un point de terminaison d'application mobile afin qu'il reçoive des notifications d'une rubrique 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.

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 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 de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour de plus amples informations sur l'API, consultez S'abonner dans AWS SDK for JavaScript Référence de l'API.

Abonner une file d'attente SQS à une rubrique

L'exemple de code suivant montre comment abonner une file d'attente Amazon SQS afin qu'elle reçoive des notifications d'une rubrique 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.

```
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;
```

```
};
```

- Pour de plus amples informations sur l'API, consultez S'abonner dans Référence de l'API AWS SDK for JavaScript .

Abonner une adresse e-mail à une rubrique

L'exemple de code suivant montre comment abonner une adresse e-mail à une rubrique 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.

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'
  // }
};
```

- Pour de plus amples informations, consultez le AWS SDK for JavaScript Guide du développeur.
- Pour de plus amples informations sur l'API, consultez S'abonner dans AWS SDK for JavaScript Référence de l'API.

S'abonner avec un filtre à une rubrique

L'exemple de code suivant montre comment s'abonner avec un filtre à une rubrique 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.

```
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 de plus amples informations sur l'API, consultez S'abonner dans Référence de l'API AWS SDK for JavaScript .

Scénarios

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 flux de travail.

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

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Le code précédent fournit les dépendances nécessaires et démarre le flux de travail. 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.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
```

```
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
```

```
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
```

```
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });
```

```
      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });
```

```
    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });
```

```
    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
```

```
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour Amazon SQS

Les exemples de code suivants montrent 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 listez 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 for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Modifier la visibilité du délai d'expiration des messages

L'exemple de code suivant montre comment modifier la visibilité du délai d'expiration d'un message 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.

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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });
```

```
// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des AWS SDK for JavaScript API.

Configuration d'une file d'attente de lettres mortes

L'exemple de code suivant montre comment configurer une file d'attente de lettres mortes dans 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.

```
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 for JavaScript API.

Créer une file d'attente

L'exemple de code suivant montre comment créer une file d'attente 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.

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 de longues interrogations.

```
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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

Créez une file d'attente Amazon SQS qui attend l'arrivée d'un message.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Success", data.QueueUrl);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section Référence des AWS SDK for JavaScript API.

Supprimer un lot de messages d'une file

L'exemple de code suivant montre comment supprimer un lot de messages d'une file d'attente 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.

```
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 for JavaScript API.

Supprimer un message d'une file

L'exemple de code suivant montre comment supprimer un message d'une file d'attente 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.

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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
```

```
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteMessageà la section Référence des AWS SDK for JavaScript API.

Suppression d'une file d'attente

L'exemple de code suivant montre comment supprimer une file d'attente 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteQueueà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};
```

```
sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteQueueà la section Référence des AWS SDK for JavaScript API.

Obtenir les attributs d'une file d'attente

L'exemple de code suivant montre comment obtenir des attributs pour une file d'attente 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.

```
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 for JavaScript API.

Obtenir l'URL d'une file d'attente

L'exemple de code suivant montre comment obtenir l'URL d'une file d'attente 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.

Obtenez 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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetQueueUrlà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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 l'URL d'une file d'attente Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous GetQueueUrlà la section Référence des AWS SDK for JavaScript API.

Répertorier les files d'attente

L'exemple de code suivant montre comment répertorier les files d'attente 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.

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);
    urls.forEach((url) => console.log(url));
  }

  return urls;
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK for JavaScript API.

Recevoir des messages depuis une file d'attente

L'exemple de code suivant montre comment recevoir des messages depuis une file d'attente 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.

Recevez un message depuis une file d'attente 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,
        })),
      }),
    );
  }
};
```

Recevez un message depuis une file d'attente Amazon SQS grâce à la prise en charge des longs sondages.

```
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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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 depuis une file d'attente Amazon SQS grâce à la prise en charge des longs sondages.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
```

```
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ReceiveMessageà la section Référence des AWS SDK for JavaScript API.

Envoyer un message à une file d'attente

L'exemple de code suivant montre comment envoyer un message à une file d'attente 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.

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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous SendMessageà la section Référence des AWS SDK for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus 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.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
```

```
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous SendMessageà la section Référence des AWS SDK for JavaScript API.

Définir les attributs de file d'attente

L'exemple de code suivant montre comment définir les attributs d'une file d'attente 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.

```
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 des interrogations longues.

```
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;
};
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des AWS SDK for JavaScript API.

Scénarios

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 flux de travail.

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

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Le code précédent fournit les dépendances nécessaires et démarre le flux de travail. 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.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
```

```
      queueUrl: response.QueueUrl,
    });

    await this.logger.log(
      MESSAGES.queueCreatedNotice
        .replace("${QUEUE_NAME}", queueName)
        .replace("${QUEUE_URL}", response.QueueUrl)
        .replace("${QUEUE_ARN}", Attributes.QueueArn),
    );
  }
}

async attachQueueIamPolicies() {
  for (const [index, queue] of this.queues.entries()) {
    const policy = JSON.stringify(
      {
        Statement: [
          {
            Effect: "Allow",
            Principal: {
              Service: "sns.amazonaws.com",
            },
            Action: "sqs:SendMessage",
            Resource: queue.queueArn,
            Condition: {
              ArnEquals: {
                "aws:SourceArn": this.topicArn,
              },
            },
          },
        ],
      },
      null,
      2,
    );

    if (index !== 0) {
      this.logger.logSeparator();
    }

    await this.logger.log(MESSAGES.attachPolicyNotice);
    console.log(policy);
    const addPolicy = await this.prompter.confirm({
      message: MESSAGES.addPolicyConfirmation.replace(
```

```
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
```

```
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });
```

```
      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });
```

```
    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });
```

```
    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
```

```
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

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 Step Functions AWS SDK for JavaScript (v3).

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Lancer une exécution de machine à états

L'exemple de code suivant montre comment démarrer l'exécution d'une machine d'état Step Functions.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
  return response;
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- Pour plus de détails sur l'API, reportez-vous StartExecutionà la section Référence des AWS SDK for 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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Assumer un rôle

L'exemple de code suivant montre comment assumer un rôle auprès de AWS STS.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// Load the AWS SDK for Node.js
const AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

var roleToAssume = {
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",
  RoleSessionName: "session1",
  DurationSeconds: 900,
};
var roleCreds;

// Create the STS service object
var sts = new AWS.STS({ apiVersion: "2011-06-15" });

//Assume Role
sts.assumeRole(roleToAssume, function (err, data) {
  if (err) console.log(err, err.stack);
  else {
    roleCreds = {
      accessKeyId: data.Credentials.AccessKeyId,
      secretAccessKey: data.Credentials.SecretAccessKey,
```

```
      sessionToken: data.Credentials.SessionToken,
    };
    stsGetCallerIdentity(roleCreds);
  }
});

//Get Arn of current identity
function stsGetCallerIdentity(creds) {
  var stsParams = { credentials: creds };
  // Create STS service object
  var sts = new AWS.STS(stsParams);

  sts.getCallerIdentity({}, function (err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data.Arn);
    }
  });
}
```

- Pour plus de détails sur l'API, reportez-vous AssumeRoleà la section Référence des AWS SDK for JavaScript API.

AWS 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 for JavaScript (v3) avec AWS Support.

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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Mise en route

Bonjour AWS Support

Les exemples de code suivants montrent comment démarrer avec AWS 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.",
      );
    } else {
      throw err;
    }
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
```

```
  }
};
```

- Pour plus de détails sur l'API, reportez-vous DescribeServicesà la section Référence des AWS SDK for JavaScript API.

Rubriques

- Actions
- Scénarios

Actions

Ajouter une communication à un dossier

L'exemple de code suivant montre comment ajouter une AWS Support communication avec une pièce jointe à un dossier de 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.

```
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 for JavaScript API.

Ajouter une pièce jointe à un ensemble

L'exemple de code suivant montre comment ajouter une AWS Support pièce jointe à un ensemble de pièces jointes.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Création d'un dossier

L'exemple de code suivant montre comment créer un nouveau AWS Support dossier.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Décrire une pièce jointe

L'exemple de code suivant montre comment décrire une pièce jointe pour un AWS Support dossier.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Décrire les dossiers

L'exemple de code suivant montre comment décrire les AWS Support cas.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Décrire les communications

L'exemple de code suivant montre comment décrire les AWS Support communications relatives à un dossier.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Décrire les niveaux de gravité

L'exemple de code suivant montre comment décrire les niveaux de AWS Support gravité.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Résoudre un dossier

L'exemple de code suivant montre comment résoudre un AWS Support problème.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 for JavaScript API.

Scénarios

Démarrer avec les dossiers

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 inquirer from "inquirer";

// Retry an asynchronous function on failure.
const retry = async ({ intervalInMs = 500, maxRetries = 10 }, fn) => {
  try {
    return await fn();
  } catch (err) {
    console.log(`Function call failed. Retrying.`);
    console.error(err.message);
    if (maxRetries === 0) throw err;
    await new Promise((resolve) => setTimeout(resolve, intervalInMs));
    return retry({ intervalInMs, maxRetries: maxRetries - 1 }, fn);
  }
};

const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};

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

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});

  try {
    await client.send(command);
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature."
      );
    } else {
      throw err;
    }
  }
};

// Get the list of available services.
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
```

```
  const { selectedService } = await inquirer.prompt({
    name: "selectedService",
    type: "list",
    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;
};

// Get the list of available support case categories for a service.
export const getCategory = async (service) => {
  const { selectedCategory } = await inquirer.prompt({
    name: "selectedCategory",
    type: "list",
    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.prompt({
    name: "selectedSeverityLevel",
    type: "list",
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

// Create a new support case and return the caseId.
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;
};

// Get an attachment set.
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.prompt({
    name: "shouldResolve",
    type: "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.
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 d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for JavaScript .
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases
  - DescribeCommunications
  - DescribeServices
  - DescribeSeverityLevels
  - ResolveCase

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 for 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 et dans des exemples interservices.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir une tâche spécifique en appelant plusieurs fonctions au sein d'un même service.

Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions sur la façon de configurer et d'exécuter le code en contexte.

Rubriques

- Actions

Actions

Suppression d’une tâche de transcription médicale

L'exemple de code suivant montre comment supprimer une tâche de transcription Amazon Transcribe Medical.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 { TranscribeClient } = require("@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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteMedicalTranscriptionJobà la section Référence des AWS SDK for JavaScript API.

Suppression d’une tâche de transcription

L'exemple de code suivant montre comment supprimer une tâche de transcription Amazon Transcribe.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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.

```
const { TranscribeClient } = require("@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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous DeleteTranscriptionJobà la section Référence des AWS SDK for JavaScript API.

Création d’une liste de tâches de transcription médicale

L'exemple de code suivant montre comment répertorier les tâches de transcription Amazon Transcribe Medical.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 { TranscribeClient } = require("@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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListMedicalTranscriptionJobsà la section Référence des AWS SDK for JavaScript API.

Création d'une liste de tâches de transcription

L'exemple de code suivant montre comment répertorier les tâches de transcription Amazon Transcribe.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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.

```
const { TranscribeClient } = require("@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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous ListTranscriptionJobsà la section Référence des AWS SDK for JavaScript API.

Démarrage d’une tâche de transcription médicale

L'exemple de code suivant montre comment démarrer une tâche de transcription Amazon Transcribe Medical.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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 { TranscribeClient } = require("@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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.

- Pour plus de détails sur l'API, reportez-vous StartMedicalTranscriptionJobà la section Référence des AWS SDK for JavaScript API.

Démarrage d’une tâche de transcription

L'exemple de code suivant montre comment démarrer une tâche de transcription Amazon Transcribe.

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus 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.

```
const { TranscribeClient } = require("@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 de plus amples informations, consultez le Guide du développeur AWS SDK for JavaScript.
- Pour plus de détails sur l'API, reportez-vous StartTranscriptionJobà la section Référence des AWS SDK for JavaScript API.

Exemples interservices utilisant le SDK pour JavaScript (v3)

Les exemples d'applications suivants utilisent le AWS SDK for JavaScript (v3) pour fonctionner sur plusieurs Services AWS applications.

Les exemples multiservices ciblent un niveau d'expérience avancé pour vous aider à commencer à créer des applications.

Exemples

- Créer une application Amazon Transcribe
- Créer une application de streaming Amazon Transcribe
- Créer une application pour soumettre des données à une table DynamoDB
- Créez un chatbot Amazon Lex pour engager les visiteurs de votre site Web
- Création d’une application de gestion des ressources photographiques permettant aux utilisateurs de gérer les photos à l’aide d’étiquettes
- Créer une application web pour suivre les données DynamoDB
- Créer un outil de suivi des éléments de travail sans serveur Aurora

- Créer une application Amazon Textract Explorer
- Créez une application qui analyse les commentaires des clients et synthétise le son
- Détectez le PPE dans les images avec Amazon Rekognition à l'aide d'un SDK AWS
- Détectez des objets dans des images avec Amazon Rekognition à l'aide d'un SDK AWS
- Détectez les personnes et les objets dans une vidéo avec Amazon Rekognition à l'aide d'un SDK AWS
- Invoquer une fonction Lambda à partir d'un navigateur
- Utiliser API Gateway pour appeler une fonction Lambda
- Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda
- Utilisent des événements planifiés pour appeler une fonction Lambda

Créer une application Amazon Transcribe

SDK pour JavaScript (v3)

Créez une application qui utilise Amazon Transcribe pour transcrire et afficher des enregistrements vocaux dans le navigateur. L'application utilise deux compartiments Amazon Simple Storage Service (Amazon S3), l'un pour héberger le code de l'application, l'autre pour stocker les transcriptions. L'application utilise un groupe d'utilisateurs Amazon Cognito pour authentifier vos utilisateurs. Les utilisateurs authentifiés disposent des autorisations AWS Identity and Access Management (IAM) nécessaires pour accéder aux services requis. AWS

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 for JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- Amazon Cognito Identity
- Amazon S3
- Amazon Transcribe

Créer une application de streaming Amazon Transcribe

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 une application pour soumettre des données à une table DynamoDB

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 for JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SNS

Créez un chatbot Amazon Lex pour engager les visiteurs de votre site Web

SDK pour JavaScript (v3)

Montre comment utiliser l'API Amazon Lex pour créer un Chatbot au sein d'une application Web afin d'engager 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 for JavaScript développeur.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Création d’une application de gestion des ressources photographiques permettant aux utilisateurs de gérer les 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 web pour suivre les données DynamoDB

SDK pour JavaScript (v3)

Montre comment utiliser l'API Amazon DynamoDB pour créer une application web dynamique qui suit les données de travail DynamoDB.

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
- Amazon SES

Créer un outil de suivi des éléments de travail sans serveur Aurora

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK for 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

Créer une application Amazon Textract Explorer

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK for 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 qui analyse les commentaires des clients et synthétise le son

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 for 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

Détectez le PPE dans les images avec Amazon Rekognition à l'aide d'un SDK AWS

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Rekognition AWS SDK for JavaScript pour créer une application permettant de détecter les équipements de protection individuelle (EPI) sur des images situées dans un compartiment Amazon Simple Storage Service (Amazon S3). L'application enregistre les résultats dans une table Amazon DynamoDB et 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'EPI à l'aide d'Amazon Rekognition.
- Vérifier une adresse e-mail pour Amazon SES.
- Mettre à jour une table DynamoDB avec les résultats.
- 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

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Détectez des objets dans des images avec Amazon Rekognition à l'aide d'un SDK AWS

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Rekognition AWS SDK for 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

Détectez les personnes et les objets dans une vidéo avec Amazon Rekognition à l'aide d'un SDK AWS

SDK pour JavaScript (v3)

Montre comment utiliser Amazon Rekognition pour créer une application permettant de détecter AWS SDK for JavaScript des visages et des objets dans des vidéos 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'EPI 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

Invoquer une fonction Lambda à partir d’un navigateur

SDK pour JavaScript (v2)

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.

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

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 for 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

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 for JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB

- Lambda
- Amazon SNS

Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda

SDK pour JavaScript (v3)

Montre comment créer un flux de travail AWS sans serveur en utilisant AWS Step Functions et le AWS SDK for JavaScript. Chaque étape du flux de travail est implémentée à l'aide d'une AWS Lambda fonction.

Lambda est un service de calcul qui vous permet d'exécuter du code sans devoir approvisionner ou gérer des serveurs. Step Functions est un service d'orchestration sans serveur qui vous permet de combiner des fonctions Lambda et d'autres services AWS afin de créer des applications métier essentielles.

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 for JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Utilisent des événements planifiés pour appeler une fonction Lambda

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 for JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

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 des identités 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 TLS minimale

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 en savoir plus 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 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 AWS Identity and Access Management (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.
- Utilisez le protocole SSL/TLS pour communiquer avec les ressources. AWS Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail.
- 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 140-2 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour en savoir plus sur les points de terminaison FIPS (Federal Information Processing Standard) disponibles, consultez Federal Information Processing Standard (FIPS) 140-2 (Normes de traitement de l’information fédérale).

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 Name (Nom). Cela inclut lorsque vous travaillez avec ce AWS produit ou service ou avec un autre produit Services AWS à l'aide de la console, de l'API ou AWS des SDK. AWS CLI Toutes les données que vous saisissez 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 des identités et des accès

AWS Identity and Access Management (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
- Authentification par des identités
- Gestion des accès à l’aide de politiques
- Comment Services AWS travailler avec IAM
- Résolution des problèmes AWS d'identité et d'accès

Public ciblé

La façon dont vous utilisez AWS Identity and Access Management (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. En comprenant 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 (connecté à AWS) en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en AWS tant qu'identité fédérée en utilisant les informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez avec une identité fédérée, votre administrateur aura précédemment configuré une fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS l'aide de la fédération, vous assumez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter au portail AWS Management Console ou au portail AWS d'accès. Pour plus d'informations sur la connexion à AWS, consultez la section Comment vous connecter à votre compte Compte AWS dans le guide de Connexion à AWS l'utilisateur.

Si vous y accédez AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes à l'aide de vos informations d'identification. Si vous n'utilisez pas d' AWS outils, vous devez signer vous-même les demandes. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer vous-même les demandes, consultez la section Signature des demandes AWS d'API dans le guide de l'utilisateur IAM.

Quelle que soit la méthode d’authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, il vous AWS recommande d'utiliser l'authentification multifactorielle (MFA) pour renforcer la sécurité de votre compte. Pour en savoir plus, veuillez consulter Multi-factor authentication (Authentification multifactorielle) dans le Guide de l'utilisateur AWS IAM Identity Center et Utilisation de l'authentification multifactorielle (MFA) dans l'interface AWS dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une identité de connexion unique qui donne un accès complet à toutes Services AWS les ressources du compte. Cette identité est

appelée utilisateur Compte AWS root et est accessible en vous connectant avec l'adresse e-mail et le mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur racine pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur racine et utilisez-les pour effectuer les tâches que seul l'utilisateur racine peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur root, consultez Tâches nécessitant des informations d'identification d'utilisateur root dans le Guide de l'utilisateur IAM.

Identité fédérée

La meilleure pratique consiste à obliger les utilisateurs humains, y compris ceux qui ont besoin d'un accès administrateur, à utiliser la fédération avec un fournisseur d'identité pour accéder à l'aide Services AWS d'informations d'identification temporaires.

Une identité fédérée est un utilisateur de l'annuaire des utilisateurs de votre entreprise, d'un fournisseur d'identité Web AWS Directory Service, du répertoire Identity Center ou de tout utilisateur qui y accède à l'aide des informations d'identification fournies Services AWS par le biais d'une source d'identité. Lorsque des identités fédérées y accèdent Comptes AWS, elles 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. Vous pouvez créer des utilisateurs et des groupes dans IAM Identity Center, ou vous pouvez vous connecter et synchroniser avec un ensemble d'utilisateurs et de groupes dans votre propre source d'identité afin de les utiliser dans toutes vos applications Comptes AWS et applications. Pour obtenir des informations sur IAM Identity Center, 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 au sein de vous Compte AWS qui possède des autorisations spécifiques pour une seule personne ou une seule application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs IAM ayant des informations d'identification à long terme tels que les clés d'accès. Toutefois, si certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons de faire pivoter les clés d'accès. Pour plus d'informations, consultez Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification dans le Guide de l'utilisateur IAM.

Un groupe IAM est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez avoir un groupe nommé IAMAdmins et accorder à ce groupe les autorisations d'administrer des ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour en savoir plus, consultez Quand créer un utilisateur IAM (au lieu d'un rôle) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité au sein de vous Compte AWS dotée d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Vous pouvez assumer temporairement un rôle IAM dans le en AWS Management Console changeant de rôle. Vous pouvez assumer un rôle en appelant une opération d' AWS API AWS CLI ou en utilisant une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez Utilisation de rôles IAM dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré – Pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, veuillez consulter la rubrique Jeux d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .
- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.
- Accès intercompte : vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, dans certains Services AWS cas, vous pouvez associer une politique directement à une ressource (au lieu d'utiliser un rôle comme

proxy). Pour en savoir plus sur la différence entre les rôles et les politiques basées sur les ressources pour l'accès intercompte, consultez Différence entre les rôles IAM et les politiques basées sur les ressources dans le Guide de l'utilisateur IAM.

- Accès multiservices — Certains Services AWS utilisent des fonctionnalités dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, une fonction de service ou un rôle lié au service.
  - Sessions d'accès direct (FAS) : lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains services, l'action que vous effectuez est susceptible de lancer une autre action dans un autre service. FAS utilise les autorisations du principal appelant et Service AWS, associées Service AWS à la demande, pour adresser des demandes aux services en aval. Les demandes FAS ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur la politique relative à la transmission de demandes FAS, consultez Sessions de transmission d'accès.
  - Fonction du service : il s'agit d'un rôle IAM attribué à un service afin de réaliser des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer une fonction du service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.
  - Rôle lié à un service — Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés au service apparaissent dans votre Compte AWS fichier et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.
- Applications exécutées sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer les informations d'identification temporaires pour les applications qui s'exécutent sur une instance EC2 et qui envoient des demandes d'API. AWS CLI AWS Cette solution est préférable au stockage des clés d'accès au sein de l'instance EC2. Pour attribuer un AWS rôle à une instance EC2 et le mettre à la disposition de toutes ses applications, vous devez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes qui s'exécutent sur l'instance EC2 d'obtenir des informations d'identification temporaires. Pour plus d'informations,

consultez Utilisation d'un rôle IAM pour accorder des autorisations à des applications s'exécutant sur des instances Amazon EC2 dans le Guide de l'utilisateur IAM.

Pour savoir dans quel cas utiliser des rôles ou des utilisateurs IAM, consultez Quand créer un rôle IAM (au lieu d'un utilisateur) dans le Guide de l'utilisateur IAM.

Gestion des 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 est un objet AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit leurs autorisations. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur root ou session de rôle) fait une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations sur la structure et le contenu des documents de politique JSON, consultez Présentation des politiques JSON dans le Guide de l'utilisateur IAM.

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C’est-à-dire, 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. Pour octroyer aux utilisateurs des autorisations d’effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM. L’administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Les politiques IAM définissent les autorisations d'une action, quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposiez d’une politique qui autorise l’action `iam:GetRole`. Un utilisateur appliquant cette politique peut obtenir des informations sur le rôle à partir de AWS Management Console AWS CLI, de ou de l' AWS API.

Politiques basées sur l’identité

Les politiques basées sur l’identité sont des documents de politique d’autorisations JSON que vous pouvez attacher à une identité telle qu’un utilisateur, un groupe d’utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d’actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l’identité, consultez Création de politiques IAM dans le Guide de l’utilisateur IAM.

Les politiques basées sur l'identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les politiques gérées sont des politiques autonomes que vous pouvez associer à plusieurs utilisateurs, groupes et rôles au sein de votre Compte AWS. Les politiques gérées incluent les politiques AWS gérées et les politiques gérées par le client. Pour découvrir comment choisir entre une politique gérée et une politique 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. Des politiques basées sur les ressources sont, par exemple, les politiques de confiance de rôle IAM et des politiques 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. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

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 (ACL)

Les listes de contrôle d'accès (ACL) vérifie quels principaux (membres de compte, utilisateurs ou rôles) ont l'autorisation d'accéder à une ressource. Les listes de contrôle d'accès sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et Amazon VPC sont des exemples de services qui prennent en charge les ACL. AWS WAF Pour en savoir plus sur les listes de contrôle d'accès, consultez Présentation des listes de contrôle d'accès (ACL) dans le Guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge d'autres types de politiques moins courants. Ces types de politiques peuvent définir le nombre maximum d'autorisations qui vous sont accordées par des types de politiques plus courants.

- Limite d'autorisations : une limite d'autorisations est une fonction avancée dans laquelle vous définissez le nombre maximal d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d'autorisations pour une entité. Les autorisations qui en résultent représentent la combinaison des politiques basées sur l'identité d'une entité et de ses limites d'autorisation. Les politiques basées sur les ressources qui spécifient l'utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d'autorisations. Un refus explicite dans l'une de ces politiques remplace l'autorisation. Pour plus d'informations sur les limites d'autorisations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCP) — Les SCP sont des politiques JSON qui spécifient les autorisations maximales pour une organisation ou une unité organisationnelle (UO) dans. AWS Organizations AWS Organizations est un service permettant de regrouper et de gérer de manière centralisée Comptes AWS les multiples propriétés de votre entreprise. Si vous activez toutes les fonctions d'une organisation, vous pouvez appliquer les politiques de contrôle de service (SCP) à l'un ou à l'ensemble de vos comptes. Le SCP limite les autorisations pour les entités figurant dans les comptes des membres, y compris chacune Utilisateur racine d'un compte AWS d'entre elles. Pour plus d'informations sur les organisations et les SCP, consultez Fonctionnement des SCP dans le Guide de l'utilisateur AWS Organizations .
- politiques de séance : les politiques de séance sont des politiques avancées que vous utilisez en tant que paramètre lorsque vous créez par programmation une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de la séance obtenue sont une combinaison des politiques basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques remplace l'autorisation. Pour plus d'informations, consultez Politiques de séance dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations obtenues 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, un utilisateur doit 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 ne dispose pas des autorisations nécessaires pour transférer le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les stratégies 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 (ACL), vous pouvez utiliser ces politiques pour donner l'accès à vos ressources.

Pour en savoir plus, 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 découvrir quelle est la différence entre l'utilisation des rôles et l'utilisation des stratégies basées sur les ressources pour l'accès intercompte, consultez Différence entre les rôles IAM et les stratégies basées sur les ressources 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. AWS fournit les ressources suivantes pour faciliter la mise en conformité :

- Guides de démarrage rapide sur la sécurité et la conformité : ces guides de déploiement abordent les considérations architecturales et indiquent les étapes à suivre pour déployer des environnements de base axés sur AWS la sécurité et la conformité.
- Architecture axée sur la sécurité et la conformité HIPAA sur Amazon Web Services : ce livre blanc décrit comment les entreprises peuvent créer des applications AWS conformes à la loi HIPAA.

> **Note**
>
> Tous ne Services AWS sont pas éligibles à la loi HIPAA. Pour plus d'informations, consultez HIPAA Eligible Services Reference.

- AWS Ressources de https://aws.amazon.com/compliance/resources/ de conformité — Cette collection de classeurs et de guides peut s'appliquer à votre secteur d'activité et à votre région.
- AWS Guides de conformité destinés aux clients — Comprenez le modèle de responsabilité partagée sous l'angle de la conformité. Les guides résument les meilleures pratiques en matière

de sécurisation Services AWS et décrivent les directives relatives aux contrôles de sécurité dans de nombreux cadres (notamment le National Institute of Standards and Technology (NIST), le Payment Card Industry Security Standards Council (PCI) et l'Organisation internationale de normalisation (ISO)).

- Évaluation des ressources à l'aide des règles du guide du AWS Config développeur : le AWS Config service évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- AWS Security Hub— Cela Service AWS fournit une vue complète de votre état de sécurité interne AWS. Security Hub utilise des contrôles de sécurité pour évaluer vos ressources AWS et vérifier votre conformité par rapport aux normes et aux bonnes pratiques du secteur de la sécurité. Pour obtenir la liste des services et des contrôles pris en charge, consultez Référence des contrôles Security Hub.
- AWS Audit Manager— Cela vous Service AWS permet d'auditer en permanence votre AWS utilisation afin de simplifier la gestion des risques et la conformité aux réglementations et aux normes du secteur.

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 TLS minimale

Pour renforcer la sécurité lors de la communication avec les AWS services, configurez le AWS SDK for JavaScript pour utiliser le protocole TLS 1.2 ou version ultérieure.

> **Important**
>
> La AWS SDK for JavaScript v3 négocie automatiquement la version TLS de plus haut niveau prise en charge par un point de terminaison de service donné AWS . Vous pouvez éventuellement appliquer une version TLS minimale requise par votre application, telle que TLS 1.2 ou 1.3, mais veuillez noter que TLS 1.3 n'est pas pris en charge par certains points de terminaison du AWS service. Certains appels peuvent donc échouer si vous appliquez le protocole TLS 1.3.

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.

Vérifier et appliquer TLS dans Node.js

Lorsque vous utilisez le AWS SDK for 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. La AWS SDK for JavaScript version 3 utilise par défaut le protocole TLS 1.3 lorsqu'il est disponible, mais utilise par défaut une version inférieure 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.

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 TLS minimale à partir de la ligne de commande, vous devez utiliser Node.js version 11.0.0 ou une version 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 `httpOptions` 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(
            {
                secureProtocol: 'TLSv1_2_method'
            }
        )
    })
});
```

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 `httpOptions` 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(
            {
                secureProtocol: 'TLSv1_3_method'
            }
        )
    })
});
```

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 *`Options Internet`*.
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 pour activer les protocoles SSL. Si vous utilisez Safari version 7 ou supérieure, le protocole TLS 1.2 est automatiquement activé.

Migrer vers la version 3

La section explique comment migrer de la version 2 à la version 3 du AWS SDK for JavaScript.

Migrez votre code vers le SDK pour V3 JavaScript

AWS SDK for 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 for JavaScript GitHub dépôt.

Pour tirer pleinement parti de la AWS SDK for 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 for JavaScript (v2) pour utiliser les API v3. 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 });
client.listTables({}, (err, data) => {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

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 });
client.listTables({}, (err, data) => {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

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.

Historique du document pour AWS SDK for JavaScript la version 3

Historique du document

Le tableau suivant décrit les modifications importantes apportées à la version V3 AWS SDK for 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
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 vers les AWS SDK et le guide de référence des outils.	22 août 2022
Appliquer une version minimale de TLS	Ajout d'informations sur le protocole TLS 1.3.	31 mars 2022
AWS Lambda Tutoriel actualisé	Ajout d'un didacticiel expliquant comment créer une application basée sur un navigateur pour envoyer des données à une table Amazon DynamoDB.	20 octobre 2020
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	20 octobre 2020

	dans Node.js pour AWS SDK for JavaScript V3.	
Migrer vers la V3	Ajout d'une rubrique pour décrire comment migrer vers la AWS SDK for JavaScript V3.	20 octobre 2020
Commencer	Rubriques mises à jour pour démarrer dans le navigateur et utiliser Node.js pour AWS SDK for 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 for 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 for 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 for 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 for 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 for 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 for JavaScript version 3.	20 octobre 2020
Exemples de services Amazon DynamoDB mis à jour	Exemples de services Amazon DynamoDB mis à jour pour la version 3. AWS SDK for JavaScript	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 for 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 for JavaScript V3.	20 octobre 2020
AWS SDK for JavaScript Aperçu du guide du développeur V3	Publication de la version préliminaire du guide du développeur AWS SDK for JavaScript V3.	19 octobre 2020