Manuel du développeur pour le kit SDK version 2

AWS SDK for JavaScript

AWS SDK for JavaScript: Manuel du développeur pour le kit SDK version 2

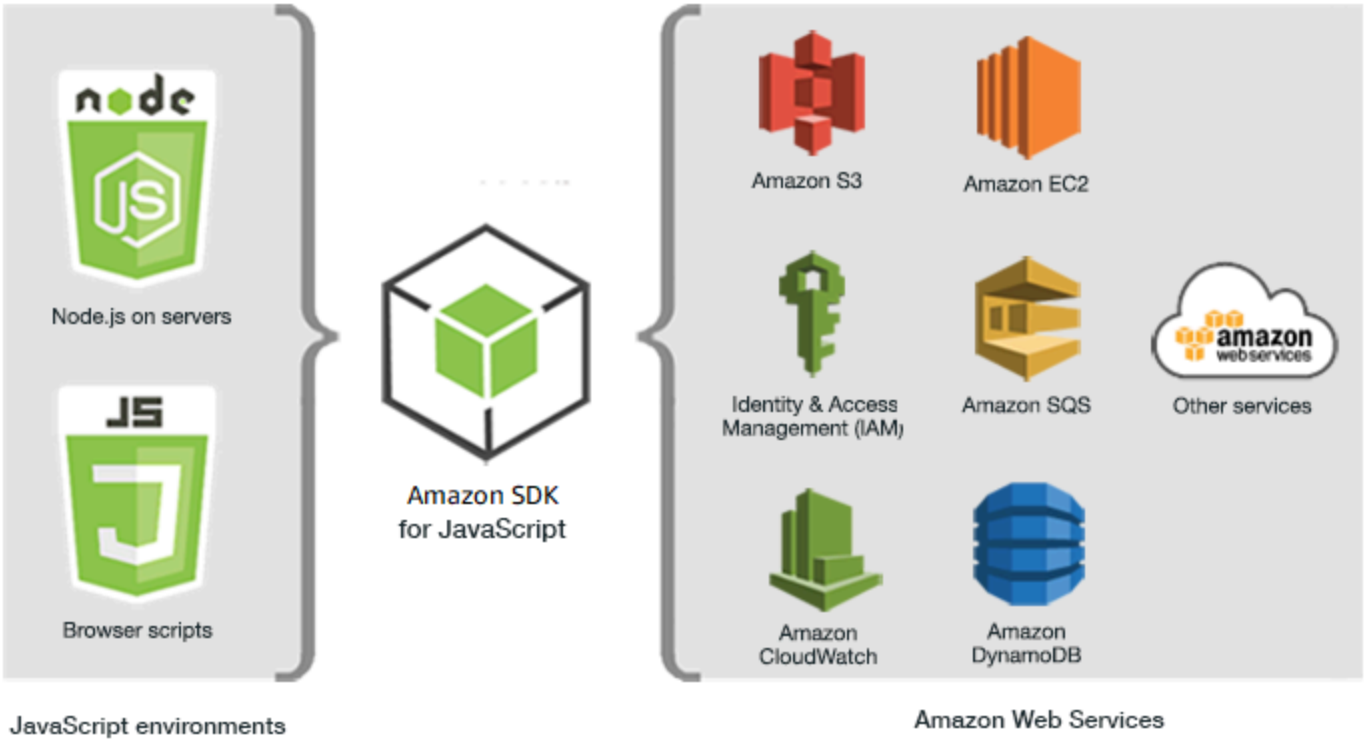
Table of Contents

Nous avons annoncé la sortie end-of-support de la AWS SDK for JavaScript v2. Nous vous recommandons de migrer vers la AWS SDK for JavaScript version 3. Pour les dates, les détails supplémentaires et les informations sur la façon de migrer, reportez-vous à l'annonce associée.

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.

Présentation d'AWS SDK for JavaScript

AWS SDK for JavaScriptfournit 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.



Tous les services ne sont pas immédiatement disponibles dans le kit SDK. Pour savoir quels services sont actuellement pris en charge par leAWS SDK for JavaScript, consultez https://github.com/aws/aws-sdk-js /blob/master/services.md. Pour plus d'informations sur le SDK pour JavaScript on GitHub, consultezRessources supplémentaires.

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

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 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, ces différences sont indiquées.

Utilisation du SDK avec AWS Cloud9

Vous pouvez également développer des applications Node.js à l'aide du SDK pour JavaScript dans l'AWS Cloud9IDE. Pour un exemple d'utilisation AWS Cloud9 pour le développement de Node.js, voir Node.js Sample for AWS Cloud9 dans le guide de AWS Cloud9 l'utilisateur. Pour plus d'informations sur AWS Cloud9 l'utilisation du SDK pour JavaScript, consultezUtilisation d'AWS Cloud9 avec l'AWS SDK for JavaScript.

Utilisation du SDK avec Amplify AWS

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

> **Note**
>
> Les frameworks tels qu'AWSAmplify peuvent ne pas offrir le même support de navigateur que le SDK pour. JavaScript Pour plus d'informations, consultez la documentation d'une infrastructure.

Utilisation du kit SDK avec des navigateurs web

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

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, ces différences sont indiquées.

Pour obtenir une liste des navigateurs pris en charge par le kit AWS SDK for JavaScript, consultez Navigateurs web pris en charge.

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 la bibliothèque d'exemples de AWS code.

Commencer avec le AWS SDK for JavaScript

AWS SDK for JavaScript Permet d'accéder aux services Web dans des scripts de navigateur ou dans le fichier Node.js. Cette section contient deux exercices de démarrage qui vous montrent comment utiliser le SDK JavaScript dans chacun de ces JavaScript environnements.

Vous pouvez également développer des applications Node.js à l'aide du SDK pour JavaScript l' AWS Cloud9 IDE. Pour un exemple d'utilisation AWS Cloud9 pour le développement de Node.js, voir Node.js Sample for AWS Cloud9 dans le guide de AWS Cloud9 l'utilisateur.

Rubriques

Démarrage dans un script de navigateur

Cet exemple de script de navigateur vous montre :

- Comment accéder aux AWS services à partir d'un script de navigateur à l'aide d'Amazon Cognito Identity.
- Comment transformer du texte en discours synthétisé à l'aide d'Amazon Polly.
- Comment utiliser un objet Presigner pour créer une URL présignée.

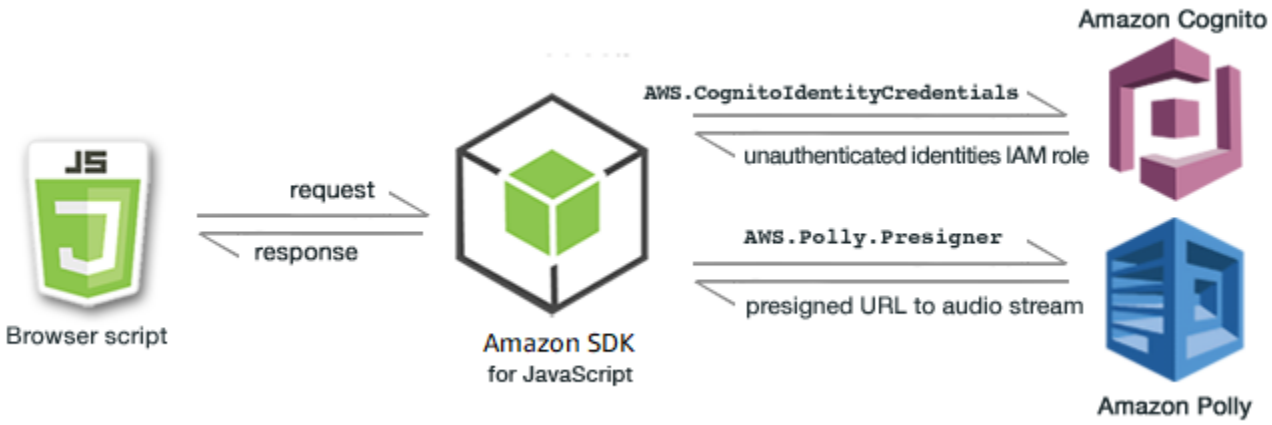
Scénario

Amazon Polly est un service cloud qui convertit le texte en un enregistrement audio réaliste. Vous pouvez utiliser Amazon Polly pour développer des applications qui améliorent l'engagement et l'accessibilité. Amazon Polly prend en charge plusieurs langues et inclut une variété de voix réalistes. Pour plus d'informations sur Amazon Polly, consultez le guide du développeur Amazon Polly.

L'exemple montre comment configurer et exécuter un script de navigateur simple qui prend le texte que vous saisissez, l'envoie à Amazon Polly, puis renvoie l'URL de l'audio synthétisé du texte pour

que vous puissiez le lire. Le script de navigateur utilise Amazon Cognito Identity pour fournir les informations d'identification nécessaires pour accéder AWS aux services. Vous verrez les modèles de base pour le chargement et l'utilisation du SDK JavaScript dans les scripts de navigateur.

> **Note**
>
> La lecture de la voix de synthèse dans cet exemple doit s'effectuer dans un navigateur prenant en charge les contenus audio HTML 5.



Le script de navigateur utilise le SDK pour synthétiser du texte JavaScript à l'aide des API suivantes :

- Constructeur `AWS.CognitoIdentityCredentials`
- Constructeur `AWS.Polly.Presigner`
- `getSynthesizeSpeechUrl`

Étape 1 : créer un pool d'identités Amazon Cognito

Dans cet exercice, vous allez créer et utiliser un pool d'identités Amazon Cognito afin de fournir un accès non authentifié au script de votre navigateur pour le service Amazon Polly. La création d'un pool d'identités crée également deux rôles IAM, l'un pour prendre en charge les utilisateurs authentifiés par un fournisseur d'identité et l'autre pour prendre en charge les utilisateurs invités non authentifiés.

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

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 : Écrire le script de navigateur

Après avoir créé votre pool d'identités Amazon Cognito, vous êtes prêt à ajouter les autorisations requises par le script de votre navigateur pour Amazon Polly.

Étape 2 : ajouter une politique au rôle IAM créé

Pour activer l'accès par script de navigateur à Amazon Polly pour la synthèse vocale, utilisez le rôle IAM non authentifié créé pour votre pool d'identités Amazon Cognito. Cela nécessite que vous ajoutiez une politique IAM au rôle. Pour plus d'informations sur 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 Polly au rôle IAM associé aux utilisateurs non authentifiés

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la 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 correspondant à AmazonPollyReadOnly.

> **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 Polly à votre rôle IAM pour les utilisateurs non authentifiés, vous êtes prêt à créer la page Web et le script de navigateur.

Étape 3 : Créer une page HTML

L'exemple d'application est constitué d'une seule page HTML qui contient l'interface utilisateur et le script de navigateur. Pour commencer, créez un document HTML et copiez-y le contenu suivant. La page comprend un champ de saisie et un bouton, un élément `<audio>` permettant de lire la synthèse vocale et un élément `<p>` permettant d'afficher des messages. (Notez que l'exemple complet est présenté au bas de cette page.)

Pour plus d'informations sur l'élément `<audio>`, consultez la section audio.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>AWS SDK for JavaScript - Browser Getting Started Application</title>
  </head>

  <body>
    <div id="textToSynth">
      <input autofocus size="23" type="text" id="textEntry" value="It's very good to
 meet you."/>
```

```
      <button class="btn default" onClick="speakText()">Synthesize</button>
      <p id="result">Enter text above then click Synthesize</p>
    </div>
    <audio id="audioPlayback" controls>
      <source id="audioSource" type="audio/mp3" src="">
    </audio>
    <!-- (script elements go here) -->
 </body>
</html>
```

Enregistrez le fichier HTML en le nommant `polly.html`. Une fois que vous avez créé l'interface utilisateur de l'application, vous êtes prêt à ajouter le code du script de navigateur qui exécute l'application.

Étape 4 : Écrire le script de navigateur

La première chose à faire lors de la création du script de navigateur est d'inclure le SDK pour JavaScript en ajoutant un `<script>` élément après l'`<audio>`élément dans la page. Pour trouver le SDK_VERSION_NUMBER actuel, consultez la référence d'API du SDK pour le guide de référence des API. JavaScript AWS SDK for JavaScript

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></script>
```

Ensuite, ajoutez un nouvel élément `<script type="text/javascript">` après l'entrée du kit SDK. Vous allez ajouter le script de navigateur à cet élément. Définissez la AWS région et les informations d'identification du SDK. Créez ensuite une fonction nommée `speakText()` qui sera appelée en tant que gestionnaire d'événements via le bouton.

Pour synthétiser la parole avec Amazon Polly, vous devez fournir divers paramètres, notamment le format sonore de la sortie, le taux d'échantillonnage, l'identifiant de la voix à utiliser et le texte à lire. Lorsque vous créez initialement les paramètres, définissez le paramètre `Text:` sur une chaîne vide. Le paramètre `Text:` sera défini sur la valeur que vous récupérez à partir de l'élément `<input>` dans la page web. Remplacez *IDENTITY_POOL_ID* et *REGION* dans le code suivant par les valeurs indiquées dans. Étape 1 : créer un pool d'identités Amazon Cognito

```
    <script type="text/javascript">

        // Initialize the Amazon Cognito credentials provider
        AWS.config.region = 'REGION';
```

```
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({IdentityPoolId:
'IDENTITY_POOL_ID'});

        // Function invoked by button click
        function speakText() {
            // Create the JSON parameters for getSynthesizeSpeechUrl
            var speechParams = {
                OutputFormat: "mp3",
                SampleRate: "16000",
                Text: "",
                TextType: "text",
                VoiceId: "Matthew"
            };
            speechParams.Text = document.getElementById("textEntry").value;
```

Amazon Polly renvoie le discours synthétisé sous forme de flux audio. Le moyen le plus simple de diffuser cet audio dans un navigateur est de demander à Amazon Polly de le rendre disponible sur une URL présignée que vous pouvez ensuite définir comme `src` attribut de l'`<audio>`élément sur la page Web.

Créez un nouvel objet de service `AWS.Polly`. Créez ensuite l'objet `AWS.Polly.Presigner` que vous utiliserez pour créer l'URL présignée qui permettra de récupérer le contenu audio de la synthèse vocale. Vous devez transmettre les paramètres vocaux que vous avez définis, ainsi que l'objet de service `AWS.Polly` que vous avez créé, au constructeur `AWS.Polly.Presigner`.

Une fois que vous avez créé l'objet Presigner, appelez la méthode `getSynthesizeSpeechUrl` de cet objet, en transmettant les paramètres vocaux. En cas de réussite, cette méthode renvoie l'URL de la synthèse vocale, que vous attribuez ensuite à l'élément `<audio>` pour la lecture.

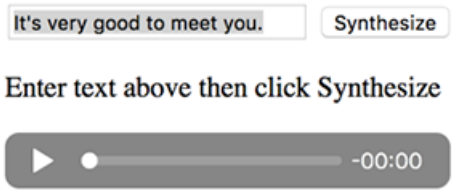
```
            // Create the Polly service object and presigner object
            var polly = new AWS.Polly({apiVersion: '2016-06-10'});
            var signer = new AWS.Polly.Presigner(speechParams, polly)

            // Create presigned URL of synthesized speech file
            signer.getSynthesizeSpeechUrl(speechParams, function(error, url) {
            if (error) {
                document.getElementById('result').innerHTML = error;
            } else {
                document.getElementById('audioSource').src = url;
                document.getElementById('audioPlayback').load();
                document.getElementById('result').innerHTML = "Speech ready to play.";
            }
```

```
        });
      }
    </script>
```

Étape 5 : Exécuter l'exemple

Pour exécuter l'exemple d'application, chargez `polly.html` dans un navigateur web. La présentation du navigateur doit ressembler à ce qui suit.



Entrez la phrase dont vous souhaitez obtenir une synthèse vocale dans la zone d'entrée, puis choisissez Synthesize (Synthétiser). Lorsque la fonction audio est prête à lire, un message s'affiche. Utilisez les commandes du lecteur audio pour écouter la synthèse vocale.

Exemple complet

Voici l'intégralité de la page HTML avec le script de navigateur. Il est également disponible ici sur GitHub.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>AWS SDK for JavaScript - Browser Getting Started Application</title>
  </head>

  <body>
    <div id="textToSynth">
      <input autofocus size="23" type="text" id="textEntry" value="It's very good to
 meet you."/>
      <button class="btn default" onClick="speakText()">Synthesize</button>
      <p id="result">Enter text above then click Synthesize</p>
    </div>
    <audio id="audioPlayback" controls>
      <source id="audioSource" type="audio/mp3" src="">
    </audio>
```

```
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.410.0.min.js"></script>
    <script type="text/javascript">

        // Initialize the Amazon Cognito credentials provider
        AWS.config.region = 'REGION';
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({IdentityPoolId:
 'IDENTITY_POOL_ID'});

        // Function invoked by button click
        function speakText() {
            // Create the JSON parameters for getSynthesizeSpeechUrl
            var speechParams = {
                OutputFormat: "mp3",
                SampleRate: "16000",
                Text: "",
                TextType: "text",
                VoiceId: "Matthew"
            };
            speechParams.Text = document.getElementById("textEntry").value;

            // Create the Polly service object and presigner object
            var polly = new AWS.Polly({apiVersion: '2016-06-10'});
            var signer = new AWS.Polly.Presigner(speechParams, polly)

            // Create presigned URL of synthesized speech file
            signer.getSynthesizeSpeechUrl(speechParams, function(error, url) {
            if (error) {
                document.getElementById('result').innerHTML = error;
            } else {
                document.getElementById('audioSource').src = url;
                document.getElementById('audioPlayback').load();
                document.getElementById('result').innerHTML = "Speech ready to play.";
            }
          });
        }
    </script>
  </body>
</html>
```

Améliorations possibles

Voici des variantes de cette application que vous pouvez utiliser pour explorer plus en détail l'utilisation du SDK JavaScript dans un script de navigateur.

- Expérimentez cet exemple avec d'autres formats de sortie audio.
- Ajoutez la possibilité de sélectionner l'une des différentes voix proposées par Amazon Polly.
- Intégrez un fournisseur d'identité tel que Facebook ou Amazon à utiliser avec le rôle IAM authentifié.

Démarrage dans Node.js

Cet exemple de code Node.js présente :

- Comment créer le manifeste `package.json` pour votre projet.
- Comment installer et inclure les modules que votre projet utilise.
- Comment créer un objet de service Amazon Simple Storage Service (Amazon S3) à partir de `AWS.S3` la classe client.
- Comment créer un compartiment Amazon S3 et y charger un objet.

Scénario

L'exemple montre comment configurer et exécuter un module Node.js simple qui crée un compartiment Amazon S3, puis y ajoute un objet texte.

Dans la mesure où les noms de compartiment dans Amazon S3 doivent être uniques au niveau mondial, cet exemple inclut un module Node.js tiers qui génère une valeur d'ID unique que vous pouvez intégrer au nom du compartiment. Ce module supplémentaire se nomme `uuid`.

Tâches prérequises

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

- Créez un répertoire de travail pour développer votre module Node.js. Nommez ce répertoire `awsnodesample`. Notez que le répertoire doit être créé dans un emplacement qui peut être mis à jour par des applications. Par exemple, dans Windows, ne créez pas le répertoire sous « C:\Program Files ».

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js. Vous pouvez trouver des téléchargements de la version actuelle et des versions LTS de Node.js pour les différents systèmes d'exploitation à l'adresse https://nodejs.org/fr/download/current/.

Table des matières

Étape 1 : Installation du SDK et des dépendances

Vous installez le SDK pour le JavaScript package à l'aide de npm (le gestionnaire de packages Node.js).

Depuis le répertoire `awsnodesample` dans le package, saisissez la commande suivante sur la ligne de commande.

```
npm install aws-sdk
```

Cette commande installe le SDK pour JavaScript dans votre projet et est mise à jour pour `package.json` répertorier le SDK en tant que dépendance du projet. Vous pouvez trouver des informations sur ce package en recherchant « aws-sdk » sur le site internet npm.

Installez ensuite le module `uuid` dans le projet en saisissant ce qui suit sur la ligne de commande. Vous installez ainsi le module et les mises à jour `package.json`. Pour plus d'informations sur `uuid`, consultez la page du module à l'adresse https://www.npmjs.com/package/uuid.

```
npm install uuid
```

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

Pour plus d'informations sur l'installation des packages Node.js, consultez Télécharger et installer des packages localement et Créer des modules Node.js sur le site internet npm (Node.js package

manager). Pour plus d'informations sur le téléchargement et l'installation du AWS SDK for JavaScript, consultezInstallation du SDK pour JavaScript.

Étape 2 : configurer vos informations d'identification

Vous devez fournir des informations d'identification pour AWS que seuls votre compte et ses ressources soient accessibles par le SDK. Pour plus d'informations sur l'obtention de vos informations d'identification, consultez Authentification du SDK avec AWS.

Pour conserver ces informations, nous vous recommandons de créer un fichier d'informations d'identification partagé. Pour savoir comment procéder, veuillez consulter la section Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé. Votre fichier d'informations d'identification doit ressembler à l'exemple suivant.

```
[default]
aws_access_key_id = YOUR_ACCESS_KEY_ID
aws_secret_access_key = YOUR_SECRET_ACCESS_KEY
```

Vous pouvez déterminer si vous avez correctement défini vos informations d'identification en exécutant le code suivant avec Node.js :

```
var AWS = require("aws-sdk");

AWS.config.getCredentials(function(err) {
  if (err) console.log(err.stack);
  // credentials not loaded
  else {
    console.log("Access key:", AWS.config.credentials.accessKeyId);
  }
});
```

De même, si vous avez correctement défini votre région dans votre `config` fichier, vous pouvez afficher cette valeur en attribuant à la variable d'`AWS_SDK_LOAD_CONFIG`environnement une valeur quelconque et en utilisant le code suivant :

```
var AWS = require("aws-sdk");

console.log("Region: ", AWS.config.region);
```

Étape 3 : Création du Package JSON pour le projet

Une fois que vous avez créé le répertoire de projet `awsnodesample`, vous créez et ajoutez un fichier `package.json` pour conserver les métadonnées de votre projet Node.js. Pour plus de détails sur l'utilisation `package.json` dans un projet Node.js, voir Création d'un fichier package.json.

Dans le répertoire du projet, créez un fichier appelé `package.json`. Ajoutez ce contenu JSON au fichier.

```
{
  "dependencies": {},
  "name": "aws-nodejs-sample",
  "description": "A simple Node.js application illustrating usage of the SDK for
 JavaScript.",
  "version": "1.0.1",
  "main": "sample.js",
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "NAME",
  "license": "ISC"
}
```

Enregistrez le fichier. Au fur et à mesure que vous installerez les modules dont vous avez besoin, la partie `dependencies` du fichier se complétera. Vous pouvez trouver un fichier JSON qui montre un exemple de ces dépendances ici GitHub.

Étape 4 : Écrire le code Node.js

Créez un fichier nommé `sample.js` pour y placer l'exemple de code. Commencez par ajouter les appels de `require` fonction pour inclure le SDK pour JavaScript et les `uuid` modules afin que vous puissiez les utiliser.

Créez un nom de compartiment unique qui sera utilisé pour créer un compartiment Amazon S3 en ajoutant une valeur d'identifiant unique à un préfixe reconnaissable, dans ce cas. `'node-sdk-sample-'` Vous générez l'ID unique en appelant le module `uuid`. Créez ensuite un nom pour le paramètre `Key` utilisé pour charger un objet dans le compartiment.

Créez un objet `promise` pour appeler la méthode `createBucket` de l'objet de service `AWS.S3`. En cas de réponse positive, créez les paramètres requis pour charger du texte dans le compartiment

que vous venez de créer. En utilisant un autre objet promise, appelez la méthode `putObject` pour charger l'objet de texte dans le compartiment.

```
// Load the SDK and UUID
var AWS = require("aws-sdk");
var uuid = require("uuid");

// Create unique bucket name
var bucketName = "node-sdk-sample-" + uuid.v4();
// Create name for uploaded object key
var keyName = "hello_world.txt";

// Create a promise on S3 service object
var bucketPromise = new AWS.S3({ apiVersion: "2006-03-01" })
  .createBucket({ Bucket: bucketName })
  .promise();

// Handle promise fulfilled/rejected states
bucketPromise
  .then(function (data) {
    // Create params for putObject call
    var objectParams = {
      Bucket: bucketName,
      Key: keyName,
      Body: "Hello World!",
    };
    // Create object upload promise
    var uploadPromise = new AWS.S3({ apiVersion: "2006-03-01" })
      .putObject(objectParams)
      .promise();
    uploadPromise.then(function (data) {
      console.log(
        "Successfully uploaded data to " + bucketName + "/" + keyName
      );
    });
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Cet exemple de code se trouve ici sur GitHub.

Étape 5 : Exécuter l'exemple

Pour exécuter l'exemple, tapez la commande suivante.

```
node sample.js
```

Si le chargement aboutit, vous verrez un message de confirmation sur la ligne de commande. Vous pouvez également trouver le compartiment et l'objet de texte téléchargé dans la console Amazon S3.

Utilisation d'AWS Cloud9 avec l'AWS SDK for JavaScript

Vous pouvez utiliserAWS Cloud9avec leAWS SDK for JavaScriptpour écrire et exécuter votre JavaScript dans le code du navigateur (ainsi que pour écrire, exécuter et déboguer votre code Node.js) à l'aide d'un simple navigateur.AWS Cloud9inclut des outils tels qu'un éditeur de code et un terminal, ainsi qu'un débogueur pour le code Node.js. L'IDE AWS Cloud9 étant basé sur le cloud, vous pouvez travailler sur vos projets depuis votre bureau, votre domicile ou tout appareil connecté à Internet. Pour obtenir des informations générales surAWS Cloud9, consultezAWS Cloud9Guide de l'utilisateur.

Suivez ces étapes pour configurerAWS Cloud9avec le kit SDK pour JavaScript :

Table des matières

Étape 1 : Configuration de votreAWSUtilisation du compte à utiliserAWS Cloud9

Commencez à utiliserAWS Cloud9en vous connectant àAWS Cloud9console en tant que consoleAWS Identity and Access Management(IAM) (par exemple, un utilisateur IAM) dotée des autorisations d'accès àAWS Cloud9dans votreAWS.

Pour configurer une entité IAM dans votreAWSaccéder à un compteAWS Cloud9, et pour vous connecter auAWS Cloud9console, consultezConfiguration d'une équipe pourAWS Cloud9dans leAWS Cloud9Guide de l'utilisateur.

Étape 2 : Configuration de votreAWS Cloud9Environnement de développement

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

VoirCréation d'un environnement dansAWS Cloud9dans leAWS Cloud9Guide de l'utilisateurpour de plus amples informations.

> **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 Cloud9pour créer un environnement, lancez une instance Amazon EC2, puis connectez la nouvelle instance au nouvel environnement. C'est le moyen le plus rapide de commencer à utiliser AWS Cloud9.

Étape 3 : Configuration du kit SDK pour JavaScript

AprèsAWS Cloud9ouvre l'IDE pour votre environnement de développement, suivez l'une des procédures suivantes ou les deux pour utiliser l'IDE pour configurer le kit SDK pour JavaScript dans votre environnement.

Pour configurer le kit SDK pour 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 pour utiliser npm pour installer le kit SDK pour JavaScript.

   ```
   npm install aws-sdk
   ```

   Si l'IDE ne parvient pas à trouver npm, exécutez les commandes suivantes, l'une après l'autre, dans le même ordre, pour installer 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 permet pas de contrôler le code suivant. Avant de l'exécuter, vérifiez son authenticité et son intégrité. Vous trouverez plus d'informations sur ce code dans le référentiel GitHub nvm.

```
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 kit SDK pour JavaScript dans le navigateur

Vous n'avez pas besoin d'installer le kit SDK pour JavaScript pour l'utiliser dans les scripts de navigateur. Vous pouvez charger le package du kit SDK hébergé pour JavaScript directement depuisAWSavec un script dans vos pages HTML.

Vous pouvez télécharger les versions distribuables réduites et non réduites du kit SDK actuel pour JavaScript depuis GitHub à l'adressehttps://github.com/aws/aws-sdk-js/tree/master/dist.

Étape 4 : Téléchargement de l'exemple de code

Utilisez le terminal que vous avez ouvert à l'étape précédente pour télécharger l'exemple de code pour le kit SDK pour JavaScript dans le kit SDK pour JavaScriptAWS Cloud9environnement de 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 le kit officielAWSDocumentation du kit 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 leEnvironnementpour ouvrir la fenêtre*ENVIRONMENT_NAME*`\aws-doc-sdk-examples\javascript\example_code`où,*NOM_ENVIRONNEMENT*est le nom de votreAWS Cloud9environnement de développement.

Pour apprendre à utiliser ces exemples et d'autres exemples de code, consultezExemples de code SDK pour JavaScript.

Étape 5 : Exécution et débogage l'exemple de code

Pour exécuter du code dans votreAWS Cloud9Environnement de développement, consultezExécuter votre codedans leAWS Cloud9Guide de l'utilisateur.

Pour déboguer le code Node.js, consultezDéboguer votre codedans leAWS Cloud9Guide de l'utilisateur.

Configuration du SDK pour JavaScript

Les rubriques de cette section expliquent comment installer le SDK JavaScript pour une utilisation dans les navigateurs Web et avec Node.js. Elles expliquent également comment charger le kit SDK afin que vous puissiez accéder aux services web pris en charge par ce dernier.

> **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

Avant d'utiliser le AWS SDK for JavaScript, déterminez si votre code doit être exécuté dans Node.js ou dans un navigateur Web. Ensuite, procédez comme suit :

- Pour Node.js, installez Node.js sur vos serveurs se cela n'est pas déjà fait.
- Pour les navigateurs web, identifiez les versions de navigateur nécessaires à la prise en charge.

Rubriques

Configuration d'un environnement AWS Node.js

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

- Choisissez une Amazon Machine Image (AMI) avec Node.js préinstallé et créez 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 de Node.js (32 bits ou 64 bits) préinstallée.
- 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, consultezTutoriel : Configuration 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

Le SDK pour JavaScript prend en charge tous les navigateurs Web modernes, y compris les versions minimales suivantes :

Navigateur	Version
Google Chrome	28.0+
Mozilla Firefox	26.0+
Opera	17.0+
Microsoft Edge	25.10+
Windows Internet Explorer	N/A
Apple Safari	5+
Navigateur Android	4.3+

> **Note**
>
> Les frameworks tels qu' AWS Amplify peuvent ne pas offrir le même support de navigateur que le SDK pour. JavaScript Pour plus d'informations, consultez la documentation d'une infrastructure.

Installation du SDK pour JavaScript

Le mode d'installation AWS SDK for JavaScript dépend de l'exécution du code dans les modules Node.js ou dans les scripts du navigateur.

Tous les services ne sont pas immédiatement disponibles dans le kit SDK. Pour savoir quels services sont actuellement pris en charge par le AWS SDK for JavaScript, consultez https://github.com/aws/aws-sdk-js /blob/master/services.md

Node

La méthode préférée pour installer AWS SDK for JavaScript for Node.js consiste à utiliser npm, le gestionnaire de packages Node.js. Pour ce faire, saisissez ce qui suit dans la ligne de commande.

```
npm install aws-sdk
```

Si le message d'erreur suivant s'affiche :

```
npm WARN deprecated node-uuid@1.4.8: Use uuid module instead
```

Saisissez les commandes ci-dessous dans la ligne de commande :

```
npm uninstall --save node-uuid
npm install --save uuid
```

Browser

Vous n'avez pas besoin d'installer le kit SDK pour l'utiliser dans les scripts de navigateur. Vous pouvez charger le package SDK hébergé directement depuis Amazon Web Services à l'aide d'un script dans vos pages HTML. Le package SDK hébergé prend en charge le sous-ensemble de AWS services qui appliquent le partage de ressources entre origines (CORS). Pour de plus amples informations, veuillez consulter Chargement du SDK pour JavaScript.

Vous pouvez créer une version de génération personnalisée du kit SDK dans laquelle vous sélectionnez les services et les versions web spécifiques que vous souhaitez utiliser. Ensuite, vous pouvez télécharger votre package de kit SDK personnalisé pour le développement en local et l'héberger pour qu'il soit utilisé par votre application. Pour plus d'informations sur la création d'une version de génération personnalisée du kit SDK, consultez Création du kit SDK pour les navigateurs.

Vous pouvez télécharger des versions distribuables minifiées et non minifiées de la version actuelle à l'adresse suivante : AWS SDK for JavaScript GitHub

https://github.com/aws/ aws-sdk-js /tree/master/dist

Installation à l'aide de Bower

Bower est un gestionnaire de package pour le web. Après avoir installé Bower, vous pouvez l'utiliser pour installer le kit SDK. Pour installer le kit SDK à l'aide de Bower, saisissez ce qui suit dans une fenêtre de terminal :

```
bower install aws-sdk-js
```

Chargement du SDK pour JavaScript

Le mode de chargement du SDK JavaScript varie selon que vous le chargez pour l'exécuter dans un navigateur Web ou dans Node.js.

Tous les services ne sont pas immédiatement disponibles dans le kit SDK. Pour savoir quels services sont actuellement pris en charge par le AWS SDK for JavaScript, consultez https://github.com/aws/ aws-sdk-js /blob/master/services.md

Node.js

Après avoir installé le SDK, vous pouvez charger le AWS package dans l'application de votre nœud à l'aide `require` de.

```
var AWS = require('aws-sdk');
```

React Native

Pour utiliser le kit SDK dans un projet React Native, installez tout d'abord le kit SDK avec npm :

```
npm install aws-sdk
```

Dans votre application, utilisez le code suivant pour référencer la version compatible React Native du kit SDK :

```
var AWS = require('aws-sdk/dist/aws-sdk-react-native');
```

Browser

Le moyen le plus rapide de démarrer avec le SDK est de charger le package du SDK hébergé directement depuis Amazon Web Services. Pour ce faire, ajoutez un élément `<script>` à vos pages HTML sous la forme suivante :

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></
script>
```

Pour trouver le SDK_VERSION_NUMBER actuel, consultez la référence d'API du SDK pour le guide de référence des API. JavaScript AWS SDK for JavaScript

Après avoir chargé le kit SDK dans votre page, ce dernier est accessible à partir de la variable globale `AWS` (ou `window.AWS`).

Si vous regroupez vos dépendances de code et de module à l'aide de browserify, vous chargez le kit SDK à l'aide de `require`, comme vous le feriez dans Node.js.

Mise à niveau du SDK à JavaScript partir de la version 1

Les remarques suivantes vous aident à mettre à niveau le SDK JavaScript de la version 1 à la version 2.

Conversion automatique en Base64 et types d'horodatage sur les entrées/sorties

Désormais, le kit SDK encode et décode automatiquement les valeurs codées en base64 ainsi que les valeurs d'horodatage, pour le compte de l'utilisateur. Cette modification affecte toutes les opérations dans lesquelles des valeurs en base64 ou d'horodatage ont été envoyées par une demande ou retournées dans une réponse qui autorise les valeurs codées en base64.

Le code utilisateur qui permettait auparavant de convertir en base64 n'est plus nécessaire. Les valeurs codées en base64 sont désormais retournées comme objets tampon à partir des réponses du serveur et peuvent également être transmises comme entrée de tampon. Par exemple, les paramètres `SQS.sendMessage` de la version 1 suivants :

```
var params = {
   MessageBody: 'Some Message',
   MessageAttributes: {
      attrName: {
         DataType: 'Binary',
         BinaryValue: new Buffer('example text').toString('base64')
      }
   }
};
```

Peuvent être réécrits comme suit.

```
var params = {
   MessageBody: 'Some Message',
   MessageAttributes: {
      attrName: {
         DataType: 'Binary',
         BinaryValue: 'example text'
      }
   }
};
```

Voici comment le message est lu.

```
sqs.receiveMessage(params, function(err, data) {
  // buf is <Buffer 65 78 61 6d 70 6c 65 20 74 65 78 74>
  var buf = data.Messages[0].MessageAttributes.attrName.BinaryValue;
  console.log(buf.toString()); // "example text"
});
```

Response.data a été déplacé. RequestId à Response.RequestId

Désormais, le kit SDK stocke les ID de demande de tous les services dans un emplacement cohérent sur l'objet `response`, plutôt que dans la propriété `response.data`. Cela permet d'améliorer la cohérence entre les services qui exposent les ID de demande de différentes façons. Il s'agit

également d'une modification avec rupture qui renomme la propriété `response.data.RequestId` en `response.requestId` (`this.requestId` dans une fonction de rappel).

Dans votre code, modifiez les éléments suivants :

```
svc.operation(params, function (err, data) {
  console.log('Request ID:', data.RequestId);
});
```

Jusqu'à ce qui suit :

```
svc.operation(params, function () {
  console.log('Request ID:', this.requestId);
});
```

Éléments d'encapsuleur exposés

Si vous utilisez `AWS.ElastiCache`, `AWS.RDS` ou `AWS.Redshift` pour certaines opérations, vous devez accéder à la réponse via la propriété de sortie de niveau supérieur dans la réponse.

Par exemple, la méthode `RDS.describeEngineDefaultParameters` utilisée pour retourner ce qui suit.

```
{ Parameters: [ ... ] }
```

Désormais, elle retourne ce qui suit.

```
{ EngineDefaults: { Parameters: [ ... ] } }
```

La liste des opérations concernés pour chaque service est affichée dans le tableau suivant.

Classe client	Opérations

`AWS.ElastiCache`	`authorizeCacheSecurityGroupIngress`
	`createCacheCluster`
	`createCacheParameterGroup`

Classe client	Opérations
	createCacheSecurityGroup createCacheSubnetGroup createReplicationGroup deleteCacheCluster deleteReplicationGroup describeEngineDefaultParameters modifyCacheCluster modifyCacheSubnetGroup modifyReplicationGroup purchaseReservedCacheNodesOffering rebootCacheCluster revokeCacheSecurityGroupIngress

Classe client	Opérations
AWS.RDS	addSourceIdentifierToSubscr iption authorizeDBSecurityGroupIngress copyDBSnapshot createDBInstance createDBInstanceReadReplica createDBParameterGroup createDBSecurityGroup createDBSnapshot createDBSubnetGroup createEventSubscription createOptionGroup deleteDBInstance deleteDBSnapshot deleteEventSubscription describeEngineDefaultParameters modifyDBInstance modifyDBSubnetGroup modifyEventSubscription modifyOptionGroup promoteReadReplica purchaseReservedDBInstances Offering

Classe client	Opérations

	`rebootDBInstance`
	
	`removeSourceIdentifierFromSubscription`
	
	`restoreDBInstanceFromDBSnapshot`
	
	`restoreDBInstanceToPointInTime`
	
	`revokeDBSecurityGroupIngress`

Classe client	Opérations
AWS.Redshift	authorizeClusterSecurityGroupIngress authorizeSnapshotAccess copyClusterSnapshot createCluster createClusterParameterGroup createClusterSecurityGroup createClusterSnapshot createClusterSubnetGroup createEventSubscription createHsmClientCertificate createHsmConfiguration deleteCluster deleteClusterSnapshot describeDefaultClusterParameters disableSnapshotCopy enableSnapshotCopy modifyCluster modifyClusterSubnetGroup modifyEventSubscription modifySnapshotCopyRetentionPeriod

Classe client	Opérations

	`purchaseReservedNodeOffering`
	
	`rebootCluster`
	
	`restoreFromClusterSnapshot`
	
	`revokeClusterSecurityGroupIngress`
	
	`revokeSnapshotAccess`
	
	`rotateEncryptionKey`

Propriétés client abandonnées

Les propriétés `.client` et `.Client` ont été supprimées des objets de service. Si vous utilisez la propriété `.Client` sur une classe de service ou la propriété `.client` sur une instance d'objet de service, supprimez ces propriétés à partir de votre code.

Le code suivant utilisé avec la version 1 du SDK pour JavaScript :

```
var sts = new AWS.STS.Client();
// or
var sts = new AWS.STS();

sts.client.operation(...);
```

Doit être remplacé par le code suivant.

```
var sts = new AWS.STS();
sts.operation(...)
```

Configuration du 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. Au minimum, vous devez configurer les options suivantes :

- La région dans laquelle vous allez demander des services.
- Les informations d'identification qui vous autorisent à accéder aux ressources du kit SDK.

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 différentes manières de configurer le SDK JavaScript pour Node.js et de l' JavaScript exécuter dans un navigateur Web.

Rubriques

Utilisation de l'objet de configuration globale

Le kit SDK peut être configuré de deux façons :

- En définissant la configuration globale à l'aide d'`AWS.Config`.
- En transmettant des informations de configuration supplémentaires à un objet de service.

Il est souvent plus facile, pour commencer, de définir la configuration globale avec `AWS.Config`, mais la configuration au niveau du service permet souvent de mieux contrôler les services individuels. La configuration globale spécifiée par `AWS.Config` fournit les paramètres par défaut des objets de service que vous créez par la suite, ce qui simplifie leur configuration. Cependant, vous pouvez mettre à jour la configuration des objets de service individuels si la configuration globale ne répond pas à vos besoins.

Paramétrage de la configuration globale

Une fois que vous avez chargé le package `aws-sdk` dans votre code, vous pouvez utiliser la variable globale `AWS` pour accéder aux classes du kit SDK et interagir avec les services individuels. Le kit SDK comprend un objet de configuration globale, `AWS.Config`, que vous utilisez pour spécifier les paramètres de configuration du kit SDK requis par votre application.

Configurez le kit SDK en définissant les propriétés d'`AWS.Config` en fonction des besoins de votre application. Le tableau suivant récapitule les propriétés `AWS.Config` couramment utilisées pour définir la configuration du kit SDK.

Options de configuration	Description
`credentials`	Obligatoire. Spécifie les informations d'identif ication utilisées pour déterminer l'accès aux services et aux ressources.
`region`	Obligatoire. Spécifie la région dans laquelle les demandes de services sont effectuées.
`maxRetries`	Facultatif. Spécifie le nombre maximal de fois où une certaine demande est relancée.
`logger`	Facultatif. Spécifie un objet Logger dans lequel sont écrites les informations de débogage.
`update`	Facultatif. Met à jour la configuration actuelle avec les nouvelles valeurs.

Pour plus d'informations sur l'objet de configuration, consultez `Class: AWS.Config`la référence de l'API.

Exemples de configuration globale

Vous devez définir la région et les informations d'identification dans `AWS.Config`. Vous pouvez définir ces propriétés dans le cadre du constructeur `AWS.Config`, comme illustré dans l'exemple de script de navigateur suivant :

```
var myCredentials = new
 AWS.CognitoIdentityCredentials({IdentityPoolId:'IDENTITY_POOL_ID'});
var myConfig = new AWS.Config({
  credentials: myCredentials, region: 'us-west-2'
});
```

Vous pouvez également les définir après avoir créé `AWS.Config` à l'aide de la méthode `update`, comme illustré dans l'exemple suivant qui met à jour la région :

```
myConfig = new AWS.Config();
myConfig.update({region: 'us-east-1'});
```

Vous pouvez obtenir vos informations d'identification par défaut en appelant la méthode `getCredentials` statique d'`AWS.config` :

```
var AWS = require("aws-sdk");

AWS.config.getCredentials(function(err) {
  if (err) console.log(err.stack);
  // credentials not loaded
  else {
    console.log("Access key:", AWS.config.credentials.accessKeyId);
  }
});
```

De même, si vous avez correctement défini votre région dans votre `config` fichier, vous pouvez obtenir cette valeur en définissant la variable d'`AWS_SDK_LOAD_CONFIG`environnement sur n'importe quelle valeur et en appelant la `region` propriété statique de `AWS.config` :

```
var AWS = require("aws-sdk");

console.log("Region: ", AWS.config.region);
```

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 devez créer 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. Lorsque vous définissez les valeurs de configuration sur un objet de service, le constructeur récupère toutes les valeurs de configuration utilisées par `AWS.Config`, y compris les informations d'identification.

Par exemple, si vous devez accéder à des objets Amazon EC2 dans plusieurs 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 AWS.EC2({region: 'ap-southeast-2', maxRetries: 15, apiVersion:
 '2014-10-01'});
var ec2_regionB = new AWS.EC2({region: 'us-east-1', maxRetries: 15, apiVersion:
 '2014-10-01'});
```

Vous pouvez également définir les valeurs de configuration spécifiques à un service lors de la configuration du kit SDK avec `AWS.Config`. L'objet de configuration globale prend en charge plusieurs options de configuration spécifiques aux services. Pour plus d'informations sur la configuration spécifique au service, consultez la référence `Class: AWS.Config`de l' AWS SDK for JavaScript API.

Données de configuration immuables

Les modifications apportées à la configuration globale s'appliquent aux demandes relatives à tous les objets de service nouvellement créés. Ces objets de service nouvellement créés sont d'abord configurés avec les données de configuration globale actuelles, puis avec les éventuelles options de configuration locale. Les mises à jour que vous apportez à l'objet `AWS.config` global ne s'appliquent pas aux objets de service précédemment créés.

Les objets de service existants doivent être mis à jour manuellement avec les nouvelles données de configuration. Sinon, vous devez créer et utiliser un nouvel objet de service qui comporte les nouvelles données de configuration. L'exemple suivant crée un nouvel objet de service Amazon S3 avec de nouvelles données de configuration :

```
s3 = new AWS.S3(s3.config);
```

Configuration de la AWS région

Une 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 configuration du SDK JavaScript afin que le SDK accède aux ressources de cette région. Certains services sont disponibles uniquement dans certaines régions.

Le SDK pour JavaScript ne sélectionne pas de région par défaut. Toutefois, vous pouvez définir la région à l'aide d'une variable d'environnement, un fichier `config` partagé ou l'objet de configuration globale.

Dans un constructeur de classe client

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

```
var s3 = new AWS.S3({apiVersion: '2006-03-01', region: 'us-east-1'});
```

Utilisation de l'objet de configuration globale

Pour définir la région dans votre JavaScript code, mettez à jour l'objet de configuration `AWS.Config` globale comme indiqué ici.

```
AWS.config.update({region: 'us-east-1'});
```

Pour plus d'informations sur les régions actuelles et les services disponibles dans chaque région, voir AWS Régions et points de terminaison dans le Références générales AWS.

À l'aide d'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.

À l'aide d'un fichier de configuration partagé

De même que le fichier d'informations d'identification partagé permet de stocker les informations d'identification pour qu'elles soient utilisées par le kit SDK, vous pouvez conserver les paramètres

des régions et autres paramètres de configuration dans un fichier partagé nommé `config` qui est utilisé par les kits SDK. Si la variable d'`AWS_SDK_LOAD_CONFIG`environnement a été définie sur une valeur quelconque, 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 :

- Sous Linux, macOS ou Unix : `~/.aws/config`
- Sous 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-east-1
   output=json
```

Pour plus d'informations sur l'utilisation de fichiers de configuration et d'informations d'identification partagés, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé la section Fichiers de configuration et d'identification dans le guide de l'AWS Command Line Interface utilisateur.

Ordre de priorité pour définir la région

L'ordre de priorité pour définir la région est le suivant :

- Si une région est transmise à un constructeur de classe client, cette région est utilisée. Si ce n'est pas le cas, alors...
- Si une région est définie sur l'objet de configuration globale, cette région est utilisée. Si ce n'est pas le cas, alors...
- Si la variable d'environnement `AWS_REGION` est une valeur truthy, cette région est utilisée. Si ce n'est pas le cas, alors...
- Si la variable d'environnement `AMAZON_REGION` est une valeur truthy, cette région est utilisée. Si ce n'est pas le cas, alors...
- Si la variable d'`AWS_SDK_LOAD_CONFIG`environnement est définie sur une valeur quelconque et que le fichier d'informations d'identification partagé (`~/.aws/credentials`ou le chemin indiqué par`AWS_SHARED_CREDENTIALS_FILE`) contient une région pour le profil configuré, cette région est utilisée. Si ce n'est pas le cas, alors...

- Si la variable d'`AWS_SDK_LOAD_CONFIG`environnement est définie sur une valeur quelconque et que le fichier de configuration (`~/.aws/config`ou le chemin indiqué par`AWS_CONFIG_FILE`) contient une région pour le profil configuré, cette région est utilisée.

Spécification des points de terminaison personnalisés

Les appels aux méthodes d'API du SDK pour JavaScript sont effectués aux URI des points de terminaison de service. Par défaut, ces points de terminaison sont générés à partir de la région que vous avez configurée pour votre code. Toutefois, il arrive que vous deviez spécifier un point de terminaison personnalisé pour vos appels d'API.

Format de la chaîne du point de terminaison

Les valeurs de point de terminaison doivent être une chaîne au format suivant :

`https://{service}.{region}.amazonaws.com`

Points de terminaison pour la région ap-northeast-3

La région `ap-northeast-3` au Japon n'est pas renvoyée par les API d'énumération de régions, telles que `EC2.describeRegions`. Pour définir les points de terminaison pour cette région, reproduisez le format décrit précédemment. Le point de terminaison Amazon EC2 pour cette région serait donc

`ec2.ap-northeast-3.amazonaws.com`

Points de terminaison pour MediaConvert

Vous devez créer un point de terminaison personnalisé à utiliser avec MediaConvert. Chaque compte client se voit attribuer son propre point de terminaison, que vous devez utiliser. Voici un exemple d'utilisation d'un point de terminaison personnalisé avec MediaConvert.

```
// Create MediaConvert service object using custom endpoint
var mcClient = new AWS.MediaConvert({endpoint: 'https://abcd1234.mediaconvert.us-
west-1.amazonaws.com'});

var getJobParams = {Id: 'job_ID'};

mcClient.getJob(getJobParams, function(err, data)) {
```

```
    if (err) console.log(err, err.stack); // an error occurred
    else console.log(data); // successful response
};
```

Pour obtenir le point de terminaison d'API de votre compte , consultez `MediaConvert.describeEndpoints` dans la référence d'API.

Dans le code, veillez à spécifier la même région que celle définie dans l'URI du point de terminaison personnalisé. Si les régions définies dans le code et dans l'URI du point de terminaison personnalisé diffèrent, les appels d'API échoueront.

Pour plus d'informations MediaConvert, consultez la `AWS.MediaConvert`classe dans la référence de l'API ou dans le guide de AWS Elemental MediaConvert l'utilisateur.

Authentification du SDK avec AWS

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

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

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

- Le AWS CLI, que vous utilisez pour démarrer une session de portail d' AWS accès avant d'exécuter votre application.
- `AWSconfig`Fichier partagé doté d'un `[default]` profil avec un ensemble de valeurs de configuration pouvant être référencées à partir du SDK. Pour connaître l'emplacement de ce fichier, 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 AWS services dans votre application.

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

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

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

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

Démarrer une session sur le portail AWS d'accès

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

```
aws sso login
```

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

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

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

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

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

Plus d'informations d'authentification

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

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

- Pour en savoir plus sur les bonnes pratiques, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.
- Pour créer des AWS informations d'identification à court terme, consultez la section Informations d'identification de sécurité temporaires dans le guide de l'utilisateur IAM.

- Pour en savoir plus sur les autres SDK destinés aux fournisseurs JavaScript d'informations d'identification, consultez la section Fournisseurs d'informations d'identification standardisés dans le Guide de référence AWS des SDK et des outils.

Définition des 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 globalement sur l'objet de configuration, à l'aide d'`AWS.Config`, ou 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 relatives aux informations d'identification

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.

> **Warning**
>
> Même s'il est possible de le faire, il est préférable de ne pas coder en dur les informations d'identification dans un script d'application ou de navigateur. Le codage en dur des informations d'identification présente le risque de révéler des informations sensibles.

Pour plus d'informations sur la gestion de vos clés d'accès, consultez la section Meilleures pratiques de gestion des clés AWS d'accès dans le Références générales AWS.

Rubriques

Définition d'informations d'identification dans Node.js

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. Lors de l'obtention d'informations d'identification dans Node.js, soyez prudent lorsque vous utilisez plusieurs sources (une variable d'environnement et un fichier JSON que vous chargez, par exemple). Vous pourriez en effet modifier les autorisations sous lesquelles s'exécute votre code sans vous en rendre compte.

Voici comment fournir vos informations d'identification dans l'ordre recommandé :

1. Chargé à partir de rôles AWS Identity and Access Management (IAM) pour Amazon EC2
2. Chargées à partir du fichier d'informations d'identification partagé (`~/.aws/credentials`)
3. Chargées à partir des variables d'environnement
4. Chargées à partir d'un fichier JSON sur disque
5. Autres classes de fournisseurs d'informations d'identification fournies par le SDK JavaScript

Si plusieurs sources d'informations d'identification sont disponibles pour le kit SDK, la priorité par défaut de la sélection est la suivante :

1. Informations d'identification qui sont explicitement définies par le biais du constructeur du service client

2. Variables d'environnement
3. Fichier d'informations d'identification partagé
4. Informations d'identification chargées à partir du fournisseur d'informations d'identification ECS (le cas échéant)
5. Informations d'identification obtenues à l'aide d'un processus d'identification spécifié dans le fichier de AWS configuration partagé ou dans le fichier d'informations d'identification partagé. Pour plus d'informations, consultez the section called "Informations d'identification via un processus d'informations d'identification configuré".
6. Informations d'identification chargées depuis AWS IAM à l'aide du fournisseur d'informations d'identification de l'instance Amazon EC2 (si elles sont configurées dans les métadonnées de l'instance)

Pour plus d'informations, consultez `Class: AWS.Credentials`et `Class: AWS.CredentialProviderChain`dans la référence de l'API.

> **Warning**
>
> Bien que cela soit possible, nous vous déconseillons de coder en dur vos AWS informations d'identification dans votre application. 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.

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

Rubriques

- Chargement des informations d'identification dans le fichier Node.js à partir de rôles IAM pour Amazon EC2
- Chargement des informations d'identification pour une fonction Lambda Node.js
- Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé
- Chargement des informations d'identification dans Node.js à partir des variables d'environnement
- Chargement des informations d'identification dans Node.js à partir d'un fichier JSON
- Chargement des informations d'identification dans Node.js via un processus d'informations d'identification configuré

Chargement des informations d'identification dans le fichier Node.js à partir de 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 Utilisation des rôles IAM pour les instances Amazon EC2 dans AWS le Guide de référence des SDK et des outils.

Chargement des 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.

Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé

Vous pouvez conserver vos données AWS d'identification dans un fichier partagé utilisé par les SDK et l'interface de ligne de commande. Lorsque le SDK est JavaScript chargé, il recherche automatiquement le fichier d'informations d'identification partagé, nommé « informations d'identification ». L'emplacement où vous conservez le fichier d'informations d'identification partagé dépend de votre système d'exploitation :

- Le fichier d'informations d'identification partagé sous Linux, Unix et macOS : `~/.aws/credentials`
- Le fichier d'informations d'identification partagé sous Windows : `C:\Users\USER_NAME\.aws\credentials`

Si vous n'avez pas encore de fichier d'informations d'identification partagé, consultez Authentification du SDK avec AWS. Une fois que vous avez suivi ces instructions, vous devriez voir un texte similaire à ce qui suit dans le fichier d'informations d'identification, où *<YOUR_ACCESS_KEY_ID>* est votre ID de clé d'accès et *<YOUR_SECRET_ACCESS_KEY>* votre clé d'accès secrète :

```
[default]
aws_access_key_id = <YOUR_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_SECRET_ACCESS_KEY>
```

Pour obtenir un exemple illustrant ce fichier en cours d'utilisation, consultez Démarrage dans Node.js.

L'en-tête de section `[default]` spécifie un profil par défaut et les valeurs associées pour les informations d'identification. Vous pouvez créer d'autres profils dans le même fichier de configuration partagé, chacun avec ses propres informations d'identification. L'exemple suivant illustre un fichier de configuration avec le profil par défaut et deux profils supplémentaires :

```
[default] ; default profile
aws_access_key_id = <DEFAULT_ACCESS_KEY_ID>
aws_secret_access_key = <DEFAULT_SECRET_ACCESS_KEY>

[personal-account] ; personal account profile
aws_access_key_id = <PERSONAL_ACCESS_KEY_ID>
aws_secret_access_key = <PERSONAL_SECRET_ACCESS_KEY>

[work-account] ; work account profile
aws_access_key_id = <WORK_ACCESS_KEY_ID>
aws_secret_access_key = <WORK_SECRET_ACCESS_KEY>
```

Par défaut, le kit SDK vérifie la variable d'environnement `AWS_PROFILE` afin de déterminer quel profil utiliser. Si la variable `AWS_PROFILE` n'est pas définie dans votre environnement, le kit SDK utilise les informations d'identification du profil `[default]`. Pour utiliser l'un des autres profils, définissez ou modifiez la valeur de la variable d'environnement `AWS_PROFILE`. Par exemple, étant donné le fichier de configuration illustré ci-dessus, pour utiliser les informations d'identification du compte professionnel, définissez la variable d'environnement `AWS_PROFILE` sur `work-account` (en fonction de votre système d'exploitation).

> **Note**
>
> Lorsque vous définissez des variables d'environnement, veillez à entreprendre les actions adéquates par la suite (en fonction des besoins de votre système d'exploitation) pour rendre les variables disponibles dans le shell ou l'environnement de commande.

Après avoir défini la variable d'environnement (si nécessaire), vous pouvez exécuter un JavaScript fichier qui utilise le SDK, par exemple un fichier nomméscript.js.

```
$ node script.js
```

Vous pouvez également sélectionner de façon explicite le profil utilisé par le kit SDK, en paramétrant process.env.AWS_PROFILE avant de charger le kit SDK ou en sélectionnant le fournisseur d'informations d'identification, comme illustré dans l'exemple ci-dessous :

```
var credentials = new AWS.SharedIniFileCredentials({profile: 'work-account'});
AWS.config.credentials = credentials;
```

Chargement des informations d'identification dans Node.js à partir des variables d'environnement

Le SDK détecte automatiquement les AWS informations d'identification définies sous forme de variables dans votre environnement et les utilise pour les demandes du SDK, éliminant ainsi le besoin de gérer les informations d'identification dans votre application. Les variables d'environnement que vous définissez pour fournir vos informations d'identification sont les suivantes :

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- AWS_SESSION_TOKEN

Pour plus de détails sur la définition des variables d'environnement, consultez la section Prise en charge des variables d'environnement dans le Guide de référence AWS des SDK et des outils.

Chargement des informations d'identification dans Node.js à partir d'un fichier JSON

Vous pouvez charger la configuration et les informations d'identification à partir d'un document JSON sur disque en utilisant AWS.config.loadFromPath. Le chemin d'accès spécifié est relatif

au répertoire de travail actuel de votre processus. Par exemple, pour charger des informations d'identification à partir d'un fichier `'config.json'` avec le contenu suivant :

```
{ "accessKeyId": <YOUR_ACCESS_KEY_ID>, "secretAccessKey": <YOUR_SECRET_ACCESS_KEY>,
 "region": "us-east-1" }
```

Utilisez ensuite le code suivant :

```
var AWS = require("aws-sdk");
AWS.config.loadFromPath('./config.json');
```

> **Note**
>
> Le chargement des données de configuration à partir d'un document JSON réinitialise toutes les données de configuration existantes. Ajoutez d'autres données de configuration après avoir appliqué cette technique. Le chargement des informations d'identification à partir d'un document JSON n'est pas pris en charge dans les scripts de navigateur.

Chargement des informations d'identification dans Node.js via un processus d'informations d'identification configuré

Vous pouvez sourcer les informations d'identification via une méthode qui n'est pas intégrée dans le kit SDK. Pour ce faire, spécifiez un processus d'identification dans le fichier de configuration partagé ou AWS dans le fichier d'informations d'identification partagé. Si la variable d'`AWS_SDK_LOAD_CONFIG`environnement est définie sur une valeur quelconque, le SDK préférera le processus spécifié dans le fichier de configuration au processus spécifié dans le fichier d'informations d'identification (le cas échéant).

Pour plus de détails sur la spécification d'un processus d'identification dans le fichier de AWS configuration partagé ou le fichier d'informations d'identification partagé, consultez la référence des AWS CLI commandes, en particulier les informations sur l'approvisionnement en informations d'identification à partir de processus externes.

Pour plus d'informations sur l'utilisation de la variable d'environnement `AWS_SDK_LOAD_CONFIG`, consultez the section called “À l'aide d'un fichier de configuration partagé” dans ce document.

Définition des 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 fournir vos informations d'identification dans l'ordre recommandé :

1. Utilisation d'Amazon Cognito Identity pour authentifier les utilisateurs et fournir les informations d'identification
2. Utilisation de l'identité web fédérée
3. Codées en dur dans le script

> **Warning**
>
> Nous vous déconseillons d'utiliser des informations codées en durAWSinformations 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

Utilisation d'Amazon Cognito Identity pour authentifier les utilisateurs

La méthode recommandée pour obtenirAWSles informations d'identification des scripts de votre navigateur doivent utiliser l'objet Amazon Cognito Identity Credentity,`AWS.CognitoIdentityCredentials`. Amazon Cognito permet l'authentification des utilisateurs via des 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 façon unique chaque compte utilisateur. Les identités Amazon Cognito ne sont pas des informations

d'identification. Elles sont échangées contre des informations d'identification grâce à la prise en charge de la fédération des identités web dans AWS Security Token Service (AWS STS).

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

Configuration de l'objet Amazon Cognito Informations d'identification

Si ce n'est pas déjà fait, créez un groupe d'identités à utiliser avec les scripts de votre navigateur dans leConsole Amazon Cognitoavant de configurer`AWS.CognitoIdentityCredentials`. Créez et associez les deux rôles IAM authentifiés et non authentifiés pour votre groupe d'identités.

L'identité des utilisateurs non authentifiés n'est pas vérifiée. Ce rôle convient donc pour les utilisateurs invités de votre application ou dans les cas où il n'est pas important que les identités des utilisateurs soient vérifiées. 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.

Une fois un groupe d'identités configuré avec les fournisseurs d'identité attachés, vous pouvez utiliser `AWS.CognitoIdentityCredentials` pour authentifier les utilisateurs. Pour configurer les informations d'identification de votre application afin d'utiliser `AWS.CognitoIdentityCredentials`, définissez la propriété `credentials` d'une configuration `AWS.Config` ou d'une configuration par service. L'exemple suivant utilise `AWS.Config` :

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030',
  Logins: { // optional tokens, used for authenticated login
    'graph.facebook.com': 'FBTOKEN',
    'www.amazon.com': 'AMAZONTOKEN',
    'accounts.google.com': 'GOOGLETOKEN'
  }
});
```

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 Facebook est l'un de vos fournisseurs d'identité, vous pouvez utiliser la fonction `FB.login` du kit SDK Facebook pour obtenir un jeton de fournisseur d'identité :

```
FB.login(function (response) {
  if (response.authResponse) { // logged in
    AWS.config.credentials = new AWS.CognitoIdentityCredentials({
      IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030',
      Logins: {
        'graph.facebook.com': response.authResponse.accessToken
      }
    });

    s3 = new AWS.S3; // we can now create our service object

    console.log('You are now logged in.');
  } else {
    console.log('There was a problem logging you in.');
  }
});
```

Permutation entre les utilisateurs non authentifiés et 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 comporte une identité unique dans Amazon Cognito, même s'il n'a pas été individuellement connecté et authentifié.

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, votre configuration par défaut peut se présenter comme suit :

```
// set the default config object
var creds = new AWS.CognitoIdentityCredentials({
 IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030'
});
AWS.config.credentials = creds;
```

Basculement vers un utilisateur authentifié

Lorsqu'un utilisateur non authentifié se connecte à un fournisseur d'identité et que vous avez un jeton, vous pouvez changer l'utilisateur non authentifié en utilisateur authentifié en appelant une

fonction personnalisée qui met à jour l'objet des informations 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;
}
```

Vous pouvez également créer un objet `CognitoIdentityCredentials`. Dans ce cas, vous devez réinitialiser les propriétés des informations d'identification des objets de service existants que vous avez créés. Les objets de service lisent dans la configuration globale uniquement à l'initialisation de l'objet.

Pour plus d'informations sur le`CognitoIdentityCredentials`objet, voir`AWS.CognitoIdentityCredentials`dans leAWS SDK for JavaScriptRéférence d'API.

Authentification des utilisateurs à l'aide de l'identité web fédérée

Vous pouvez configurer directement des fournisseurs d'identité individuels pour y accéder.AWSà l'aide de la fédération d'identité web.AWSprend actuellement en charge l'authentification des utilisateurs avec la fédération des identités web par le biais de plusieurs fournisseurs d'identité :

- Login with Amazon
- Connexion avec Facebook
- Connexion avec Google

Vous devez tout d'abord enregistrer votre application avec les fournisseurs pris en charge par votre application. Ensuite, créez un rôle IAM et configurez les autorisations associées. Le rôle IAM que vous créez sert ensuite à accorder les autorisations que vous avez configurées par le biais du fournisseur d'identité respectif. Par exemple, vous pouvez configurer un rôle grâce auquel les utilisateurs connectés avec Facebook ont un accès en lecture à un compartiment Amazon S3 spécifique que vous contrôlez.

Une fois que vous disposez d'un rôle IAM avec des privilèges configurés et une application enregistrée auprès des fournisseurs d'identité de votre choix, vous pouvez configurer le kit SDK pour obtenir les informations d'identification du rôle IAM à l'aide du code d'aide, comme suit :

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
   RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>/:role/<WEB_IDENTITY_ROLE_NAME>',
   ProviderId: 'graph.facebook.com|www.amazon.com', // this is null for Google
   WebIdentityToken: ACCESS_TOKEN
});
```

La valeur du paramètre `ProviderId` dépend du fournisseur d'identité spécifié. La valeur du paramètre `WebIdentityToken` est le jeton d'accès extrait d'une connexion réussie avec le fournisseur d'identité. Pour plus d'informations sur la configuration et l'extraction des jetons d'accès pour chaque fournisseur d'identité, consultez la documentation du fournisseur d'identité.

Étape 1 : Enregistrement auprès des fournisseurs d'identité

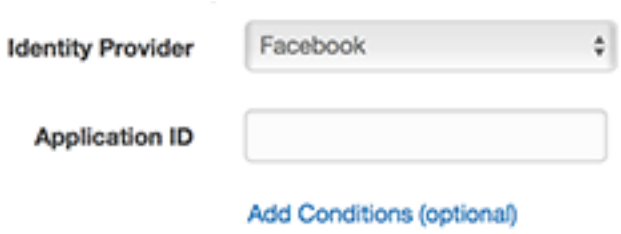
Pour commencer, enregistrez une application auprès des fournisseurs d'identité que vous choisissez de prendre en charge. Il vous sera demandé de fournir des informations qui identifient votre application et éventuellement son auteur. Les fournisseurs d'identité sauront ainsi qui reçoit leurs informations utilisateur. Dans chaque cas, le fournisseur d'identité émettra un ID d'application que vous utiliserez pour configurer des rôles utilisateur.

Étape 2 : Création d'un rôle IAM pour un fournisseur d'identité

Une fois que vous avez obtenu l'ID d'application auprès d'un fournisseur d'identité, accédez à la console IAM à l'adressehttps://console.aws.amazon.com/iam/pour créer un nouveau rôle IAM.

Pour créer un rôle IAM pour un fournisseur d'identité

1. Dans la section Rôles de la console, choisissez Créer un rôle.
2. Entrez un nom pour le nouveau rôle qui vous aidera à assurer le suivi de son utilisation, tel que **`facebookIdentity`**, puis choisissez Étape suivante.
3. Sous Sélectionner un type de rôle, choisissez Rôle pour l'accès au fournisseur d'identité.
4. Sous Octroyer l'accès aux fournisseurs d'identité Web, choisissez Sélectionner.
5. De laFournisseur d'identité, sélectionnez le fournisseur d'identité à utiliser pour ce rôle IAM.



6. Entrez l'ID d'application fourni par le fournisseur d'identité dans le champ ID d'application, puis choisissez Étape suivante.
7. Configurez les autorisations pour les ressources que vous souhaitez exposer, afin d'autoriser l'accès à des opérations spécifiques sur des ressources spécifiques. Pour plus d'informations sur les autorisations IAM, consultez.Présentation d'AWSAutorisations IAMdans leIAM User Guide. Vérifiez et, si nécessaire, personnalisez la relation d'approbation du rôle, puis choisissez Étape suivante.
8. Attachez les stratégies supplémentaires dont vous avez besoin, puis choisissez Étape suivante. Pour plus d'informations sur les stratégies IAM, consultez.Présentation des politiques IAMdans leIAM User Guide.
9. Vérifiez le nouveau rôle, puis choisissez Créer un rôle.

Vous pouvez spécifier d'autres contraintes pour le rôle, par exemple en définissant sa portée sur des ID utilisateur spécifiques. Si le rôle octroie des autorisations en écriture à vos ressources, veillez à définir correctement sa portée pour les utilisateurs dotés des privilèges appropriés. Sinon, n'importe quel utilisateur avec une identité Amazon, Facebook ou Google sera en mesure de modifier les ressources de votre application.

Pour plus d'informations sur l'utilisation de la fédération des identités web dans IAM, consultez.A propos de la fédération d'identité webdans leIAM User Guide.

Étape 3 : Obtention d'un jeton d'accès du fournisseur après connexion

Configurez l'action de connexion de votre application en utilisant le kit SDK du fournisseur d'identité. Vous pouvez télécharger et installer un kit SDK JavaScript du fournisseur d'identité qui permet aux utilisateurs de se connecter à l'aide d'OAuth ou d'OpenID. Pour plus d'informations sur la façon de télécharger et de configurer le code du kit SDK dans votre application, consultez la documentation relative au kit SDK de votre fournisseur d'identité :

- Login with Amazon
- Connexion avec Facebook

- Connexion avec Google

Étape 4 : Obtenir les informations d'identification

Après avoir configuré votre application, les rôles et les autorisations des ressources, ajoutez le code à votre application afin d'obtenir des informations d'identification temporaires. Ces informations d'identification sont fournies par le AWS Security Token Service à l'aide de la fédération des identités web. Les utilisateurs se connectent au fournisseur d'identité, qui renvoie un jeton d'accès. Configuration de l'`AWS.WebIdentityCredentials`à l'aide de l'ARN pour le rôle IAM que vous avez créé pour ce fournisseur d'identité :

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
    RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>',
    ProviderId: 'graph.facebook.com|www.amazon.com', // Omit this for Google
    WebIdentityToken: ACCESS_TOKEN // Access token from identity provider
});
```

Les objets de service créés par la suite disposeront des informations d'identification appropriées. Les objets créés avant la définition de la propriété `AWS.config.credentials` ne disposeront pas des informations d'identification actuelles.

Vous pouvez également créer la propriété `AWS.WebIdentityCredentials` avant d'extraire le jeton d'accès. Vous pourrez ainsi créer des objets de service qui dépendent des informations d'identification avant de charger le jeton d'accès. Pour ce faire, créez l'objet des informations d'identification sans le paramètre `WebIdentityToken` :

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
  RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>',
  ProviderId: 'graph.facebook.com|www.amazon.com' // Omit this for Google
});

// Create a service object
var s3 = new AWS.S3;
```

Ensuite, définissez `WebIdentityToken` dans le rappel issu du kit SDK du fournisseur d'identité qui contient le jeton d'accès :

```
AWS.config.credentials.params.WebIdentityToken = accessToken;
```

Exemples d'identité web fédérée

Voici quelques exemples d'utilisation d'identités web fédérées pour obtenir les informations d'identification dans le code JavaScript du navigateur. Exécutez ces exemples à partir d'une méthode d'hébergement http:// ou https:// afin que le fournisseur d'identité puisse les rediriger vers votre application.

Exemple de connexion Login with Amazon

Le code ci-dessous indique comment utiliser Login with Amazon comme fournisseur d'identité.

```
<a href="#" id="login">
  <img border="0" alt="Login with Amazon"
    src="https://images-na.ssl-images-amazon.com/images/G/01/lwa/
btnLWA_gold_156x32.png"
    width="156" height="32" />
</a>
<div id="amazon-root"></div>
<script type="text/javascript">
  var s3 = null;
  var clientId = 'amzn1.application-oa2-client.1234567890abcdef'; // client ID
  var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

  window.onAmazonLoginReady = function() {
    amazon.Login.setClientId(clientId); // set client ID

    document.getElementById('login').onclick = function() {
      amazon.Login.authorize({scope: 'profile'}, function(response) {
        if (!response.error) { // logged in
          AWS.config.credentials = new AWS.WebIdentityCredentials({
            RoleArn: roleArn,
            ProviderId: 'www.amazon.com',
            WebIdentityToken: response.access_token
          });

          s3 = new AWS.S3();

          console.log('You are now logged in.');
        } else {
          console.log('There was a problem logging you in.');
        }
      });
    };
```

```
  };

  (function(d) {
    var a = d.createElement('script'); a.type = 'text/javascript';
    a.async = true; a.id = 'amazon-login-sdk';
    a.src = 'https://api-cdn.amazon.com/sdk/login1.js';
    d.getElementById('amazon-root').appendChild(a);
  })(document);
</script>
```

Exemple de connexion avec Facebook

Le code ci-dessous indique comment utiliser Connexion avec Facebook comme fournisseur d'identité.

```
<button id="login">Login</button>
<div id="fb-root"></div>
<script type="text/javascript">
var s3 = null;
var appId = '1234567890'; // Facebook app ID
var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

window.fbAsyncInit = function() {
  // init the FB JS SDK
  FB.init({appId: appId});

  document.getElementById('login').onclick = function() {
    FB.login(function (response) {
      if (response.authResponse) { // logged in
        AWS.config.credentials = new AWS.WebIdentityCredentials({
          RoleArn: roleArn,
          ProviderId: 'graph.facebook.com',
          WebIdentityToken: response.authResponse.accessToken
        });

        s3 = new AWS.S3;

        console.log('You are now logged in.');
      } else {
        console.log('There was a problem logging you in.');
      }
    });
  };
```

```
};

// Load the FB JS SDK asynchronously
(function(d, s, id){
   var js, fjs = d.getElementsByTagName(s)[0];
   if (d.getElementById(id)) {return;}
   js = d.createElement(s); js.id = id;
   js.src = "//connect.facebook.net/en_US/all.js";
   fjs.parentNode.insertBefore(js, fjs);
 }(document, 'script', 'facebook-jssdk'));
</script>
```

Exemple de connexion avec Google+

Le code ci-dessous indique comment utiliser Connexion avec Google+ comme fournisseur d'identité. Le jeton d'accès utilisé pour la fédération des identités web à partir de Google est stocké dans `response.id_token`, plutôt que dans `access_token` comme pour les autres fournisseurs d'identité.

```
<span
  id="login"
  class="g-signin"
  data-height="short"
  data-callback="loginToGoogle"
  data-cookiepolicy="single_host_origin"
  data-requestvisibleactions="http://schemas.google.com/AddActivity"
  data-scope="https://www.googleapis.com/auth/plus.login">
</span>
<script type="text/javascript">
  var s3 = null;
  var clientID = '1234567890.apps.googleusercontent.com'; // Google client ID
  var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

  document.getElementById('login').setAttribute('data-clientid', clientID);
  function loginToGoogle(response) {
    if (!response.error) {
      AWS.config.credentials = new AWS.WebIdentityCredentials({
        RoleArn: roleArn, WebIdentityToken: response.id_token
      });

      s3 = new AWS.S3();

      console.log('You are now logged in.');
```

```
    } else {
      console.log('There was a problem logging you in.');
    }
  }

  (function() {
    var po = document.createElement('script'); po.type = 'text/javascript'; po.async =
true;
    po.src = 'https://apis.google.com/js/client:plusone.js';
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(po,
s);
  })();
</script>
```

Verrouillage des versions d'API

AWS les services disposent de numéros de version d'API pour assurer le suivi de la compatibilité des API. Les versions d'API dans les AWS services sont identifiées par une chaîne de date `YYYY-mm-dd` formatée. Par exemple, la version actuelle de l'API pour Amazon S3 est`2006-03-01`.

Il est recommandé de verrouiller la version de l'API pour un service si vous l'utilisez dans le code de production. Cela permet d'isoler vos applications des modifications de service résultant des mises à jour apportées au kit SDK. Si vous ne spécifiez aucune version d'API lors de la création des objets de service, le kit SDK utilise la dernière version d'API par défaut. Votre application pourrait alors référencer une API mise à jour avec des modifications nuisibles pour votre application.

Afin de verrouiller la version d'API que vous utilisez pour un service, transmettez le paramètre `apiVersion` lors de la construction de l'objet de service. Dans l'exemple ci-dessous, un objet de service `AWS.DynamoDB` nouvellement créé est verrouillé sur la version d'API `2011-12-05` :

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2011-12-05'});
```

Vous pouvez configurer globalement un ensemble de versions d'API de service en spécifiant le paramètre `apiVersions` dans `AWS.Config`. Par exemple, pour définir des versions spécifiques des API DynamoDB et Amazon EC2 en plus de l'API Amazon Redshift actuelle, configurez comme suit : `apiVersions`

```
AWS.config.apiVersions = {
  dynamodb: '2011-12-05',
  ec2: '2013-02-01',
```

```
  redshift: 'latest'
};
```

Obtention des versions d'API

Pour obtenir la version de l'API d'un service, consultez la section Verrouiller la version de l'API sur la page de référence du service, par exemple https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html pour Amazon S3.

Considérations à propos de 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)`.

Utilisation de 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.

```
var http = require('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
var fs = require('fs');
// Invoke readFile method
fs.readFile('index.html', function(err, data) {
  if (err) {
    throw err;
  } else {
    // Successful file read
  }
});
```

Pour obtenir une liste complète de tous les modules intégrés fournis par Node.js, consultez la documentation relative à Node.js v6.11.1 sur le site web de Node.js.

Utilisation de packages NPM

Outre les modules intégrés, vous pouvez également inclure et intégrer du code tiers à partir de npm, le 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 sur npm et pour obtenir 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.

Un exemple de package npm que vous pouvez utiliser avec le AWS SDK for JavaScript est`browserify`. Pour plus de détails, consultez Création du kit SDK en tant que dépendance avec Browserify. `webpack` est un autre exemple. Pour plus de détails, consultez Création d'une offre groupée d'applications avec Webpack.

Rubriques

- Configuration de maxSockets dans Node.js
- Réutilisation des connexions avec Keep-Alive dans Node.js
- Configuration de proxys pour Node.js
- Enregistrement de solutions groupées de certificats dans Node.js

Configuration de 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 ci-dessous indique comment définir `maxSockets` pour tous les objets de service que vous créez. Cet exemple autorise jusqu'à 25 connexions simultanées pour chaque point de terminaison de service.

```
var AWS = require('aws-sdk');
var https = require('https');
var agent = new https.Agent({
   maxSockets: 25
});

AWS.config.update({
   httpOptions:{
      agent: agent
   }
});
```

Le même principe s'applique également une définition par service.

```
var AWS = require('aws-sdk');
var https = require('https');
var agent = new https.Agent({
   maxSockets: 25
});

var dynamodb = new AWS.DynamoDB({
   apiVersion: '2012-08-10'
   httpOptions:{
      agent: agent
   }
});
```

Lorsque vous utilisez la valeur par défaut du paramètre `https`, le kit SDK récupère la valeur `maxSockets` de `globalAgent`. Si la valeur `maxSockets` n'est pas définie ou est `Infinity`, le kit SDK assume une valeur `maxSockets` de 50.

Pour plus d'informations sur la définition de `maxSockets` dans Node.js, consultez la documentation en ligne de Node.js.

Réutilisation des connexions avec Keep-Alive dans Node.js

Par défaut, 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, vous pouvez réutiliser une connexion existante.

Pour les opérations de courte durée de vie, telles que les requêtes DynamoDB, la surcharge de latence de la configuration d'une connexion TCP peut être supérieure à l'opération elle-même. En outre, étant donné que le chiffrement DynamoDB au repos est intégréAWS à KMS, il est possible que la base de données doive rétablir de AWS KMS nouvelles entrées de cache pour chaque opération.

Le moyen le plus simple de configurer le SDK pour JavaScript réutiliser les connexions TCP consiste à définir la variable d' `AWS_NODEJS_CONNECTION_REUSE_ENABLED`environnement sur. 1 Cette fonctionnalité a été ajoutée dans la version 2.463.0.

Vous pouvez également définir la propriété `keepAlive` d'un agent HTTP ou HTTPS définie sur `true`, comme illustré dans l'exemple suivant.

```
const AWS = require('aws-sdk');
// http or https
const http = require('http');
const agent = new http.Agent({
  keepAlive: true,
// Infinity is read as 50 sockets
  maxSockets: Infinity
});

AWS.config.update({
  httpOptions: {
    agent
  }
});
```

L'exemple suivant montre comment effectuer une configuration `keepAlive` pour un client DynamoDB uniquement :

```
const AWS = require('aws-sdk')
// http or https
const https = require('https');
const agent = new https.Agent({
  keepAlive: true
});

const dynamodb = new AWS.DynamoDB({
  httpOptions: {
    agent
  }
```

```
});
```

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

Configuration de 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, tel qu'un agent proxy. Pour installer l'agent proxy, entrez la commande ci-dessous sur la ligne de commande.

```
npm install proxy-agent --save
```

Si vous choisissez d'utiliser un autre proxy, commencez par suivre les instructions d'installation et de configuration pour ce proxy. Pour utiliser ce proxy ou un autre proxy tiers dans votre application, vous devez définir la propriété `httpOptions` d' `AWS.Config` afin de spécifier le proxy choisi. Cet exemple illustre `proxy-agent`.

```
var AWS = require("aws-sdk");
var ProxyAgent = require('proxy-agent').ProxyAgent;
AWS.config.update({
  httpOptions: { agent: new ProxyAgent('http://internal.proxy.com') }
});
```

Pour plus d'informations sur les autres bibliothèques proxy, consultez l'article relatif à npm, le gestionnaire de packages de Node.js.

Enregistrement de solutions groupées 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. L'`https.Agent` nouvellement créé est ensuite utilisé pour mettre à jour la configuration du kit SDK.

```
var fs = require('fs');
```

```
var https = require('https');
var certs = [
  fs.readFileSync('/path/to/cert.pem')
];

AWS.config.update({
  httpOptions: {
    agent: new https.Agent({
      rejectUnauthorized: true,
      ca: certs
    })
  }
});
```

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

- Création du kit SDK pour les navigateurs
- Partage des ressources cross-origine (CORS)

Création du kit SDK pour les navigateurs

Le SDK pour JavaScript est fourni sous forme de JavaScript fichier avec prise en charge d'un ensemble de services par défaut. Ce fichier est généralement chargé dans les scripts du navigateur à l'aide d'une balise `<script>` qui référence le package du kit SDK hébergé. Cependant, la prise en charge de services autres que l'ensemble par défaut peut être nécessaire ; sinon, vous devrez personnaliser le kit SDK.

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.

Rubriques

- Utilisation du générateur de SDK pour créer le SDK pour JavaScript
- Utilisation de la CLI pour créer le SDK pour JavaScript
- Création de services et de versions d'API spécifiques
- Création du kit SDK en tant que dépendance avec Browserify

Utilisation du générateur de SDK pour créer le SDK pour JavaScript

Le moyen le plus simple de créer votre propre version du AWS SDK for JavaScript est d'utiliser l'application Web SDK Builder à l'adresse https://sdk.amazonaws.com/builder/js. Utilisez le générateur de kits SDK pour spécifier des services, et leurs versions d'API, à inclure dans votre génération.

Choisissez Select all services (Sélectionner tous les services) ou Select default services (Sélectionner les services par défaut) comme point de départ à partir duquel ajouter ou supprimer des services. Choisissez Development (Développement) pour obtenir un code plus lisible ou Minified (Réduit) pour créer une génération réduite à déployer. Après avoir choisi les services et les versions à inclure, choisissez Build (Créer) pour générer et télécharger votre kit SDK personnalisé.

Utilisation de la CLI pour créer le SDK pour JavaScript

Pour créer le SDK destiné à JavaScript utiliser le AWS CLI, vous devez d'abord cloner le référentiel Git qui contient la source du SDK. Vous devez avoir installé Git et Node.js sur votre ordinateur.

Tout d'abord, clonez le dépôt depuis GitHub et changez-le de répertoire dans le répertoire :

```
git clone https://github.com/aws/aws-sdk-js.git
cd aws-sdk-js
```

Après avoir cloné le référentiel, téléchargez les modules de dépendance pour le kit SDK et l'outil de génération :

```
npm install
```

Vous pouvez désormais générer une version en package du kit SDK.

Création à partir de la ligne de commande

L'outil de génération se trouve dans `dist-tools/browser-builder.js`. Exécutez ce script en tapant :

```
node dist-tools/browser-builder.js > aws-sdk.js
```

Cette commande crée le fichier aws-sdk.js. Ce fichier est décompressé. Par défaut, ce package comprend uniquement l'ensemble de services standard.

Réduction de la sortie de génération

Pour réduire la quantité de données sur le réseau, les JavaScript fichiers peuvent être compressés par le biais d'un processus appelé minification. La minification supprime les commentaires, les espaces inutiles et d'autres caractères qui facilitent la lisibilité par les humains sans nuire à l'exécution du code. L'outil de génération peut produire une sortie non compressée ou réduite. Pour réduire votre sortie de génération, définissez la variable d'environnement `MINIFY` :

```
MINIFY=1 node dist-tools/browser-builder.js > aws-sdk.js
```

Création de services et de versions d'API spécifiques

Vous pouvez sélectionner les services à inclure dans le kit SDK. Pour sélectionner les services, spécifiez les noms de services, délimités par des virgules, comme paramètres. Par exemple, pour créer uniquement Amazon S3 et Amazon EC2, utilisez la commande suivante :

```
node dist-tools/browser-builder.js s3,ec2 > aws-sdk-s3-ec2.js
```

Vous pouvez également sélectionner des versions d'API spécifiques des générations de services en ajoutant le nom de version après le nom du service. Par exemple, pour créer les deux versions d'API d'Amazon DynamoDB, utilisez la commande suivante :

```
node dist-tools/browser-builder.js dynamodb-2011-12-05,dynamodb-2012-08-10
```

Les identificateurs de service et les versions d'API sont disponibles dans les fichiers de configuration spécifiques au service à l'adresse https://github.com/aws/ /tree/master/apis. aws-sdk-js

Création de tous les services

Vous pouvez générer tous les services et versions d'API en incluant le paramètre `all` :

```
node dist-tools/browser-builder.js all > aws-sdk-full.js
```

Création de services spécifiques

Pour personnaliser l'ensemble de services inclus dans la génération, transmettez la variable d'environnement `AWS_SERVICES` à la commande Browserify qui contient la liste des services que vous souhaitez. L'exemple suivant crée les services Amazon EC2, Amazon S3 et DynamoDB.

```
$ AWS_SERVICES=ec2,s3,dynamodb browserify index.js > browser-app.js
```

Création du kit SDK en tant que dépendance avec Browserify

Node.js comprend un mécanisme basé sur un module permettant d'inclure le code et les fonctionnalités de développeurs tiers. Cette approche modulaire n'est pas prise en charge de manière native par l' JavaScript exécution dans les navigateurs Web. Toutefois, avec l'outil Browserify, vous pouvez appliquer l'approche modulaire Node.js et utiliser les modules écrits pour Node.js dans le navigateur. Browserify crée les dépendances des modules pour un script de navigateur dans un JavaScript fichier unique et autonome que vous pouvez utiliser dans le navigateur.

Vous pouvez créer le kit SDK en tant que dépendance de bibliothèque pour n'importe quel script de navigateur en utilisant Browserify. Par exemple, le code Node.js suivant nécessite le kit SDK :

```
var AWS = require('aws-sdk');
var s3 = new AWS.S3();
s3.listBuckets(function(err, data) { console.log(err, data); });
```

Cet exemple de code peut être compilé dans une version compatible avec un navigateur en utilisant Browserify :

```
$ browserify index.js > browser-app.js
```

L'application, y compris ses dépendances du kit SDK, devient alors disponible dans le navigateur par l'intermédiaire de `browser-app.js`.

Pour plus d'informations sur Browserify, consultez le site web Browserify.

Partage des ressources cross-origine (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 environnement JavaScript utilise la sécurité CORS, vous devez configurer CORS avec le service.

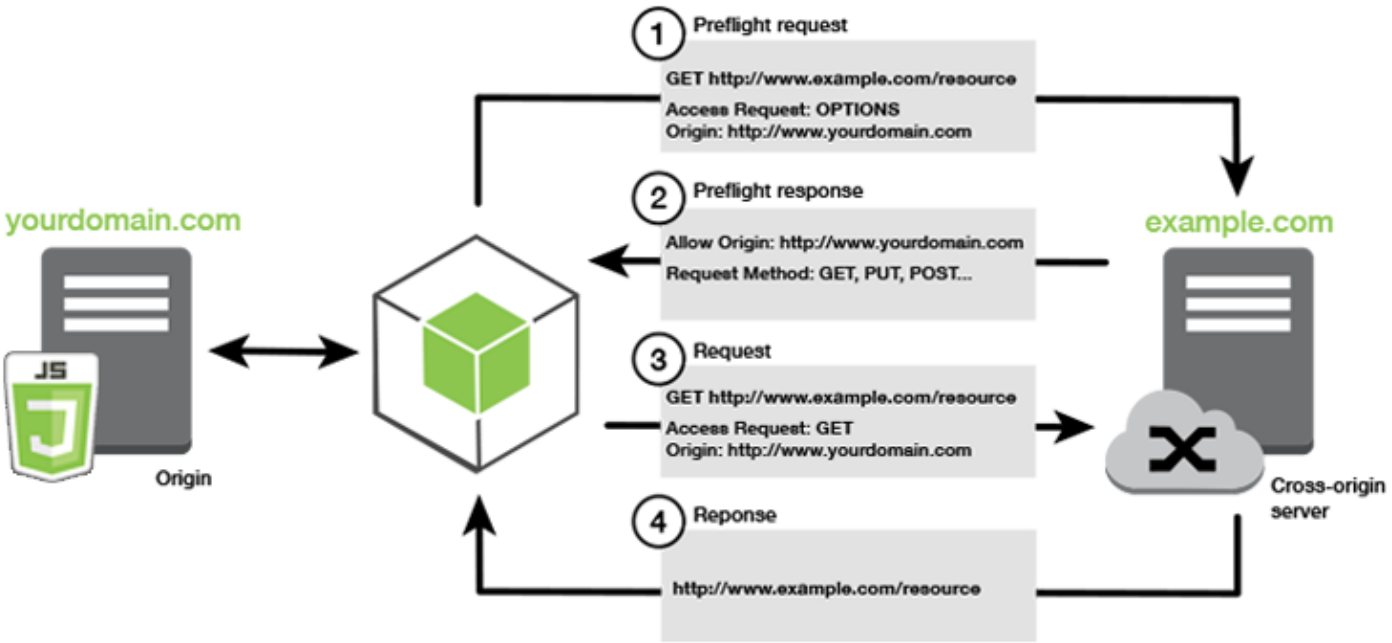
CORS détermine s'il convient d'autoriser le partage des ressources dans une demande cross-origin en se basant sur :

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

Mode de fonctionnement de 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.



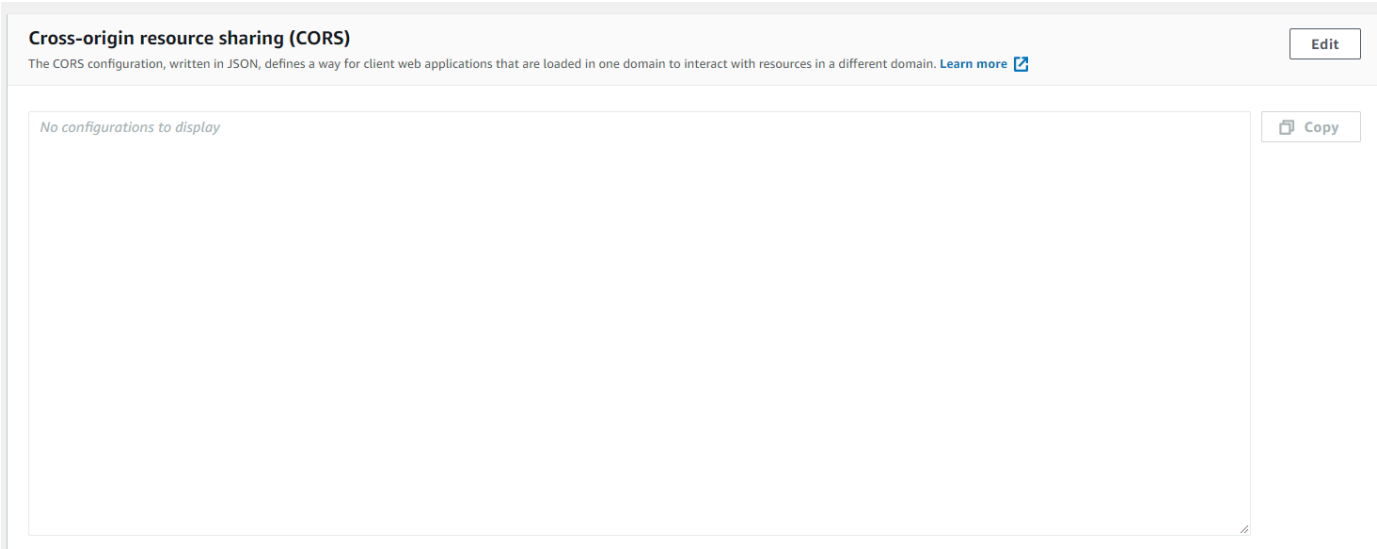
La configuration CORS est-elle obligatoire ?

Les compartiments Amazon S3 nécessitent une configuration CORS pour que vous puissiez effectuer des opérations sur ces derniers. Dans certains environnements JavaScript, CORS n'est pas appliquée et, par conséquent, la configuration CORS est inutile. Par exemple, si vous hébergez votre application à partir d'un compartiment Amazon S3 et que vous accédez à des ressources à partir de`*.s3.amazonaws.com`ou d'un autre point de terminaison spécifique, vos demandes n'accéderont 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.

Configuration de CORS pour un compartiment Amazon S3

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

1. Dans la console Amazon S3, sélectionnez le compartiment que vous souhaitez modifier.
2. Select theAutorisationset faites défiler vers le bas jusqu'à l'ongletPartage des ressources cross-origine (CORS)Volet.



3. ChoisissezModifier, et saisissez votre configuration CORS dans le champÉditeur de configuration CORS, puisEnregistrer.

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>`, qui spécifie les origines de domaine que vous autorisez à effectuer des demandes inter-domaines.
- `<AllowedMethod>`, qui spécifie un type de demande que vous autorisez (GET, PUT, POST, DELETE, HEAD) dans les demandes inter-domaines.
- `<AllowedHeader>`, qui spécifie les en-têtes autorisés dans une demande en amont.

Pour obtenir des exemples de configurations, voirComment configurer le CORS sur un compartiment ?dans leManuel de l'utilisateur d'Amazon Simple Storage Service.

Exemple de configuration CORS

L'exemple de configuration CORS suivant autorise un utilisateur à afficher, ajouter, supprimer ou mettre à jour des objets à l'intérieur d'un compartiment à partir du domaine `example.org`, mais il est recommandé de limiter la portée de `<AllowedOrigin>` au 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. Elle permet au modèle de sécurité du navigateur d'autoriser une demande à Amazon S3. Les autorisations doivent être configurées via des autorisations de compartiment ou des autorisations de rôle IAM.

Vous pouvez utiliser`ExposeHeader`pour permettre au kit SDK de lire les en-têtes de réponse renvoyés par Amazon S3. Par exemple, si vous souhaitez lire l'en-tête `ETag` à partir d'un PUT ou d'un chargement partitionné, vous devez inclure la balise `ExposeHeader` dans votre configuration, comme illustré 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.

Création d'une offre groupée d'applications avec Webpack

Les applications web des scripts de navigateur ou l'utilisation dans Node.js des modules de code créent 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 des dépendances, vous pouvez utiliser un créateur d'offre groupée de modules comme Webpack.

Le créateur d'offre groupée de modules Webpack analyse votre code d'application, en recherchant des instructions `require` ou `import` pour créer des solutions groupées contenant toutes les ressources nécessaires à votre application, afin que les ressources puissent facilement être diffusées via une page web. Le kit SDK pour JavaScript peut être inclus dans Webpack comme l'une des dépendances à inclure dans la création groupée de sortie.

Pour plus d'informations sur Webpack, consultez le créateur d'offre groupée de modules Webpack sur GitHub.

Installation de Webpack

Pour installer le créateur d'offre groupée de modules Webpack, vous devez tout d'abord installer npm, le gestionnaire de package Node.js. Saisissez la commande suivante pour installer l'interface de ligne de commande Webpack et le module JavaScript.

```
npm install webpack
```

Il se peut également que vous deviez installer un plug-in Webpack pour pouvoir charger les fichiers JSON. Saisissez la commande suivante pour installer le plug-in de chargeur JSON.

```
npm install json-loader
```

Configuration de Webpack

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

```
// 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'
  },
  module: {
    /**
      * Tell webpack how to load 'json' files.
      * When webpack encounters a 'require()' statement
      * where a 'json' file is being imported, it will use
      * the json-loader.
      */
    loaders: [
      {
        test: /\.json$/,
        loaders: ['json']
      }
    ]
  }
}
```

Dans cet exemple, `browser.js` est spécifié comme le point d'entrée. Le point d'entrée est le fichier que Webpack utilise pour rechercher des modules importés. Le nom de fichier de la sortie est spécifié en tant que `bundle.js`. Ce fichier de sortie contient tout le code JavaScript nécessaire à l'application pour s'exécuter. Si le code spécifié dans le point d'entrée importe ou nécessite d'autres modules, tels que le kit SDK pour JavaScript, ce code est regroupé sans que vous ayez besoin de le spécifier dans la configuration.

La configuration installée auparavant dans le plug-in `json-loader` indique à Webpack comment importer des fichiers JSON. Par défaut, Webpack prend uniquement en charge JavaScript, mais utilise des chargeurs pour ajouter la prise en charge de l'importation d'autres types de fichiers. Webpack génère une erreur lors de la génération de la création d'une offre groupée si le kit SDK pour JavaScript fait un usage approfondi des fichiers JSON.`json-loadern`'est pas inclus.

Exécution de Webpack

Pour créer une application permettant d'utiliser Webpack, ajoutez les éléments suivants à l'objet `scripts` dans votre fichier `package.json`.

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

Voici un exemple `package.json` qui illustre l'ajout de 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": "^2.6.1"
  },
  "devDependencies": {
    "json-loader": "^0.5.4",
    "webpack": "^1.13.2"
  }
}
```

Pour créer votre application, saisissez la commande suivante.

```
npm run build
```

Ensuite, le créateur d'offre groupée de modules Webpack génère le fichier JavaScript spécifié dans le répertoire racine de votre projet.

Utilisation de la solution groupée Webpack

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

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

```
    </body>
</html>
```

Importation de services individuels

Webpack a l'avantage d'analyser les dépendances de votre code et de regrouper uniquement le code nécessaire à votre application. Si vous utilisez le kit SDK pour JavaScript, seule la création d'une offre groupée des parties du kit SDK réellement utilisées par votre application peut réduire considérablement la taille de la sortie Webpack.

Prenons l'exemple de code suivant utilisé pour créer un objet de service Amazon S3.

```
// Import the AWS SDK
var AWS = require('aws-sdk');

// Set credentials and Region
// This can also be done directly on the service client
AWS.config.update({region: 'us-west-1', credentials: {YOUR_CREDENTIALS}});

var s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

La fonction `require()` spécifie le kit SDK entier. Un kit groupé Webpack généré avec ce code comprend le kit SDK entier, mais ce dernier n'est pas obligatoire lorsque seule la classe client Amazon S3 est utilisée. La taille de la création groupée est nettement inférieure si seule la partie du kit SDK nécessaire au service Amazon S3 est incluse. Il en est de même pour la définition de la configuration qui n'a pas besoin du kit SDK entier, car vous pour définir les données de configuration sur l'objet de service Amazon S3.

Voici à quoi ressemble le même code lorsqu'il comprend uniquement la partie Amazon S3 du kit SDK.

```
// Import the Amazon S3 service client
var S3 = require('aws-sdk/clients/s3');

// Set credentials and Region
var s3 = new S3({
    apiVersion: '2006-03-01',
    region: 'us-west-1',
    credentials: {YOUR_CREDENTIALS}
  });
```

Création d'une offre groupée pour Node.js

Vous pouvez utiliser Webpack pour générer des solutions groupées qui s'exécutent dans Node.js en le spécifiant comme cible dans la configuration.

```
target: "node"
```

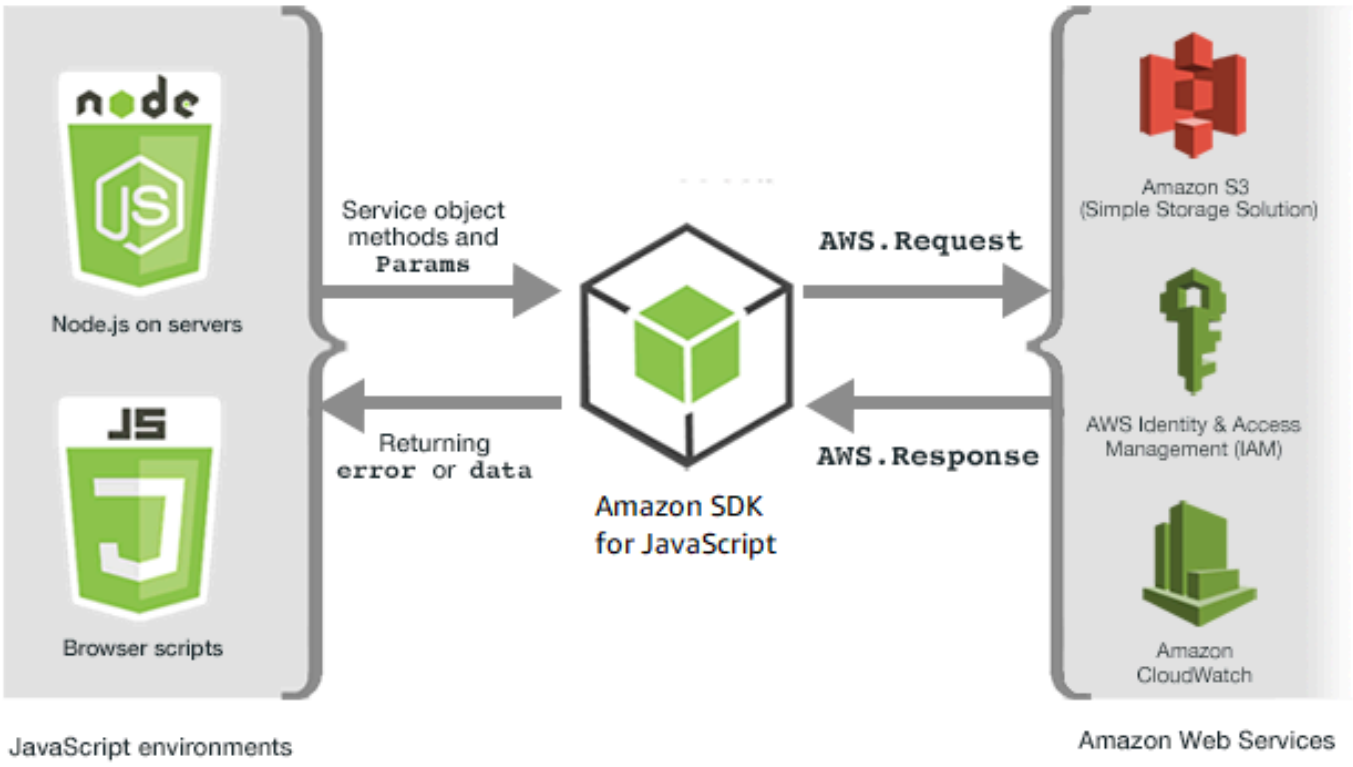
Cela s'avère utile lors de l'exécution d'une application Node.js dans un environnement où l'espace disque est limité. Voici un exemple de configuration `webpack.config.js` avec Node.js spécifié comme cible de sortie.

```
// Import path for resolving file paths
var path = require('path');
module.exports = {
  // Specify the entry point for our app
  entry: [
    path.join(__dirname, 'node.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",
  module: {
    /**
      * Tell webpack how to load JSON files.
      * When webpack encounters a 'require()' statement
      * where a JSON file is being imported, it will use
      * the json-loader
      */
    loaders: [
      {
        test: /\.json$/,
        loaders: ['json']
      }
    ]
  }
}
```

Utilisation des services du SDK pour JavaScript

Le kit AWS SDK for JavaScript permet d'accéder aux services qu'il prend en charge grâce à un ensemble de classes client. À partir de ces classes client, vous créez des objets d'interface de service, couramment appelés objets de service. Chaque service AWS pris en charge possède une ou plusieurs classes client offrant des API de bas niveau pour l'utilisation des fonctionnalités et des ressources des services. Par exemple, les API Amazon DynamoDB sont disponibles par le biais de cette classe. `AWS.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 demande est encapsulée dans le kit SDK par l'objet `AWS.Request`. La réponse est encapsulée dans le SDK par

l'`AWS.Response`objet, qui est fourni 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
- Journalisation des appels AWS SDK for JavaScript
- Appel de services de façon asynchrone
- Utilisation de l'objet de réponse
- Utilisation du format JSON

Création et appel d'objets de service

L' JavaScript API prend en charge la plupart AWS des services disponibles. Chaque classe de service de l' JavaScript API donne accès à tous les appels d'API de son 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 kit SDK dans Node.js, vous ajoutez le package du kit SDK à votre application à l'aide de `require`, qui prend en charge tous les services actuels.

```
var AWS = require('aws-sdk');
```

Lorsque vous utilisez le SDK avec un navigateur JavaScript, vous chargez le package SDK dans les scripts de votre navigateur à l'aide du package SDK hébergé par AWS. Pour charger le package du kit SDK, ajoutez l'élément `<script>` suivant :

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></script>
```

Pour trouver le SDK_VERSION_NUMBER actuel, consultez la référence d'API du SDK pour le guide de référence des API. JavaScript AWS SDK for JavaScript

Le package SDK hébergé par défaut prend en charge un sous-ensemble des services disponibles. AWS Pour obtenir la liste des services par défaut du package du kit SDK hébergé pour le navigateur, consultez Services pris en charge dans la référence d'API. Vous pouvez utiliser le kit SDK avec d'autres services si le contrôle de sécurité CORS est désactivé. Dans ce cas, vous pouvez créer une

version personnalisée du kit SDK afin d'inclure les services supplémentaires dont vous avez besoin. Pour plus d'informations sur la création d'une version personnalisée du kit SDK, consultez Création du kit SDK pour les navigateurs.

Demande de services individuels

Exiger le SDK pour, JavaScript comme indiqué précédemment, inclut le SDK complet dans votre code. Vous pouvez également choisir de demander uniquement les services individuels utilisés par votre code. Considérez le code suivant utilisé pour créer un objet de service Amazon S3.

```
// Import the AWS SDK
var AWS = require('aws-sdk');

// Set credentials and Region
// This can also be done directly on the service client
AWS.config.update({region: 'us-west-1', credentials: {YOUR_CREDENTIALS}});

var s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

Dans l'exemple précédent, la fonction `require` spécifie l'ensemble du kit SDK. La quantité de code à transporter sur le réseau ainsi que la charge de mémoire associée à votre code seraient considérablement réduites si seule la partie du SDK dont vous avez besoin pour le service Amazon S3 était incluse. Pour demander un service individuel, appelez la fonction `require` comme indiqué, en incluant le constructeur de service en lettres minuscules.

```
require('aws-sdk/clients/SERVICE');
```

Voici à quoi ressemble le code permettant de créer l'objet de service Amazon S3 précédent lorsqu'il inclut uniquement la partie Amazon S3 du SDK.

```
// Import the Amazon S3 service client
var S3 = require('aws-sdk/clients/s3');

// Set credentials and Region
var s3 = new S3({
    apiVersion: '2006-03-01',
    region: 'us-west-1',
    credentials: {YOUR_CREDENTIALS}
  });
```

Vous pouvez toujours accéder à l'espace de AWS noms global sans que tous les services y soient attachés.

```
require('aws-sdk/global');
```

Cette technique est utile lorsque vous appliquez la même configuration sur plusieurs services individuels, par exemple pour fournir les mêmes informations d'identification à tous les services. Les demandes de services individuels doivent réduire le temps de chargement et la consommation de mémoire dans Node.js. Après avoir utilisé un outil de regroupement tel que Browserify ou webpack, le fait de demander des services individuels réduit la taille du kit SDK. Cela est utile dans les environnements où la mémoire ou l'espace disque sont limités, tels qu'un appareil IoT ou une fonction Lambda.

Création d'objets de service

Pour accéder aux fonctionnalités de service par le biais de l' JavaScript API, vous devez d'abord créer un objet de service par lequel vous accédez à un ensemble de fonctionnalités fournies par la classe client sous-jacente. En général, une classe client est fournie pour chaque service. Cependant, certains services répartissent l'accès à leurs fonctionnalités entre plusieurs classes client.

Pour utiliser une fonctionnalité, vous devez créer une instance de la classe permettant d'accéder à cette fonctionnalité. L'exemple suivant montre la création d'un objet de service pour DynamoDB à partir de `AWS.DynamoDB` la classe client.

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2012-08-10'});
```

Par défaut, un objet de service est configuré avec les mêmes paramètres globaux que ceux utilisés pour configurer le kit SDK. Vous pouvez cependant configurer un objet de service avec des données de configuration d'exécution spécifiques à cet objet de service. Les données de configuration spécifiques à un service sont appliquées après les paramètres de configuration généraux.

Dans l'exemple suivant, un objet de service Amazon EC2 est créé avec une configuration pour une région spécifique, mais utilise sinon la configuration globale.

```
var ec2 = new AWS.EC2({region: 'us-west-2', apiVersion: '2014-10-01'});
```

En plus de prendre en charge une configuration spécifique à un service appliquée à un objet de service individuel, vous pouvez appliquer une configuration spécifique à un service à tous les objets de service nouvellement créés d'une classe donnée. Par exemple, pour configurer tous les objets

de service créés à partir de la classe Amazon EC2 afin d'utiliser la région USA West (Oregon) (`us-west-2`), ajoutez ce qui suit à l'objet de configuration `AWS.config` global.

```
AWS.config.ec2 = {region: 'us-west-2', apiVersion: '2016-04-01'};
```

Verrouillage de la version d'API d'un objet de service

Vous pouvez verrouiller un objet de service sur la version d'API spécifique d'un service en spécifiant l'option `apiVersion` lorsque vous créez l'objet. Dans l'exemple suivant, un objet de service DynamoDB est créé et est verrouillé selon une version d'API spécifique.

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2011-12-05'});
```

Pour plus d'informations sur le verrouillage de la version de l'API d'un objet de service, consultez Verrouillage des versions d'API.

Spécification des paramètres d'un 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 `getObject` méthode. Pour plus d'informations sur la transmission de paramètres JSON, consultez Utilisation du format JSON.

```
s3.getObject({Bucket: 'bucketName', Key: 'keyName'});
```

Pour plus d'informations sur les paramètres Amazon S3, consultez `Class: AWS.S3`la référence de l'API.

En outre, vous pouvez lier des valeurs à des paramètres individuels lorsque vous créez un objet de service à l'aide du paramètre `params`. La valeur du paramètre `params` des objets de service est une carte qui spécifie une ou plusieurs des valeurs des paramètres définies par le service. L'exemple suivant montre le `Bucket` paramètre d'un objet de service Amazon S3 lié à un compartiment nomméMyBucket`myBucket`.

```
var s3bucket = new AWS.S3({params: {Bucket: 'myBucket'}, apiVersion: '2006-03-01' });
```

En liant l'objet de service à un compartiment, l'objet de service `s3bucket` traite la valeur du paramètre `myBucket` en tant que valeur par défaut n'ayant plus besoin d'être spécifiée pour les

opérations ultérieures. Toutes les valeurs de paramètre liés sont ignorées lorsque vous utilisez l'objet pour des opérations dans lesquelles la valeur du paramètre n'est pas applicable. Vous pouvez remplacer ce paramètre lié lorsque vous effectuez des appels sur l'objet de service en spécifiant une nouvelle valeur.

```
var s3bucket = new AWS.S3({ params: {Bucket: 'myBucket'}, apiVersion: '2006-03-01' });
s3bucket.getObject({Key: 'keyName'});
// ...
s3bucket.getObject({Bucket: 'myOtherBucket', Key: 'keyOtherName'});
```

Les détails sur les paramètres disponibles pour chaque méthode sont disponibles dans la référence d'API.

Journalisation des appels AWS SDK for JavaScript

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

Pour activer l'enregistreur d'événements et enregistrer les entrées de journal dans la console, ajoutez l'instruction suivante dans votre code.

```
AWS.config.logger = console;
```

Voici un exemple de la sortie du journal.

```
[AWS s3 200 0.185s 0 retries] createMultipartUpload({ Bucket: 'js-sdk-test-bucket',
 Key: 'issues_1704' })
```

Utilisation d'un enregistreur d'événements tiers

Vous pouvez également utiliser un enregistreur d'événements tiers, à condition qu'il dispose d'opérations `write()` et `log()` pour écrire sur un fichier journal ou un serveur. Vous devez installer et configurer votre enregistreur personnalisé conformément aux instructions avant de pouvoir l'utiliser avec le SDK pour. JavaScript

logplease est un type d'enregistreur d'événements que vous pouvez utiliser dans les scripts de navigateur ou dans Node.js. Dans Node.js, vous pouvez configurer logplease pour écrire des entrées de journal dans un fichier journal. Vous pouvez également l'utiliser avec webpack.

Lorsque vous utilisez un enregistreur d'événements tiers, définissez toutes les options avant d'affecter l'enregistreur d'événements à `AWS.Config.logger`. Par exemple, le code suivant spécifie un fichier journal externe et définit le niveau de journalisation pour logplease

```
// Require AWS Node.js SDK
const AWS = require('aws-sdk')
// Require logplease
const logplease = require('logplease');
// Set external log file option
logplease.setLogfile('debug.log');
// Set log level
logplease.setLogLevel('DEBUG');
// Create logger
const logger = logplease.create('logger name');
// Assign logger to SDK
AWS.config.logger = logger;
```

Pour plus d'informations sur logplease, consultez le logger simple logger de JavaScript logplease activé. GitHub

Appel de services de façon 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 vers un service AWS, le script de navigateur continue de s'exécuter et, lors du processus, peut essayer d'exécuter du code qui dépend de ce résultat asynchrone avant d'être renvoyé.

Dans le cadre des appels asynchrones vers un service AWS, vous gérez ces appels afin que votre code ne tente pas d'utiliser les données avant qu'elles 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.

Rubriques

- Gestion des appels asynchrones
- Utilisation d'une fonction de rappel anonyme
- Utilisation d'un écouteur d'événements d'un objet de demande

- Utilisation d'async/await
- Utiliser les JavaScript promesses

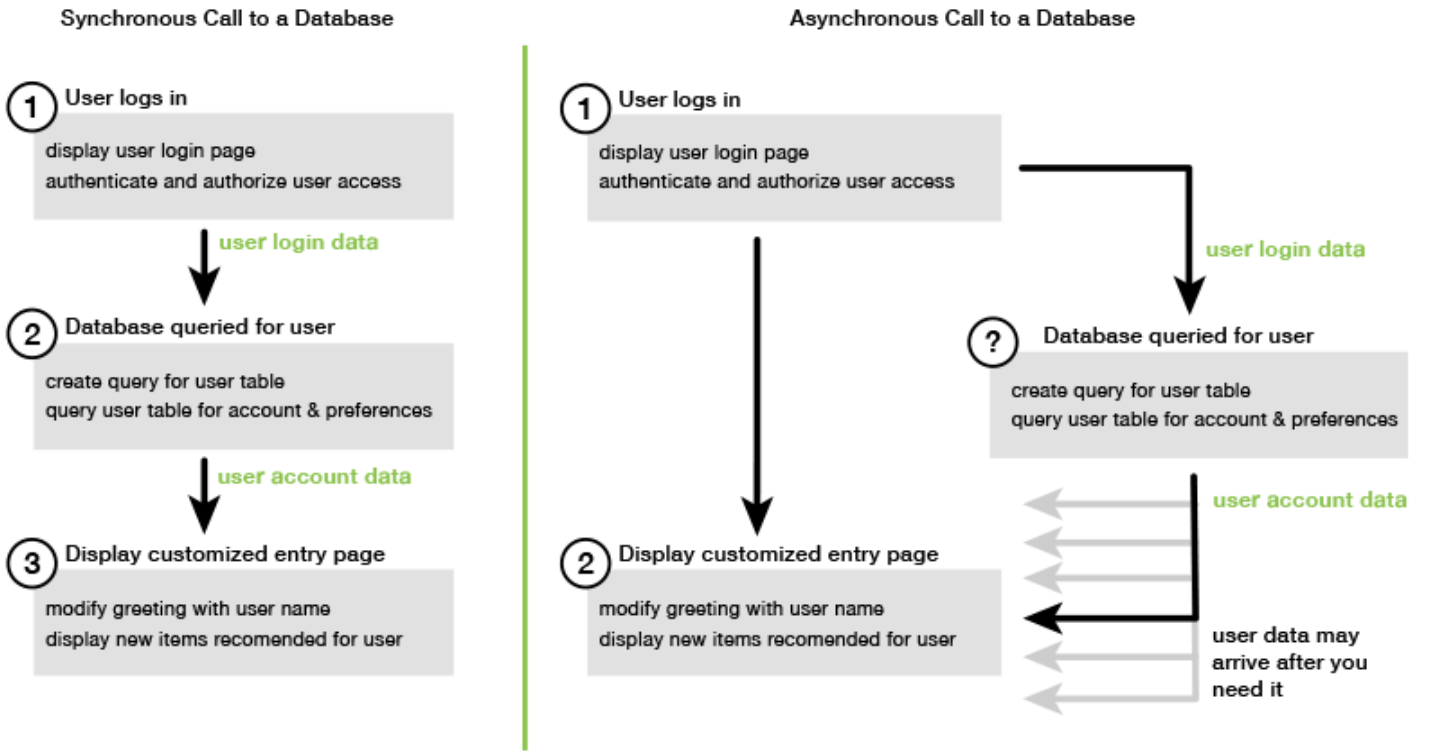
Gestion des appels asynchrones

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 pourrait donc pas se terminer avant que la base de données ait renvoyé les préférences du client. 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 empêcher le blocage du site web dans ces conditions, appelez la base de données de façon 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.



Utilisation d'une fonction de rappel anonyme

Chaque méthode d'objet de service qui crée un objet `AWS.Request` peut accepter une fonction de rappel anonyme en tant que dernier paramètre. La signature de cette fonction de rappel est :

```
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.

```
new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
  } else {
    console.log(data); // request succeeded
  }
});
```

Accès aux objets de demande et de réponse

Dans la fonction de rappel, le JavaScript mot-clé `this` fait référence à l'`AWS.Response`objet sous-jacent pour la plupart des services. Dans l'exemple suivant, la propriété `httpResponse` d'un objet `AWS.Response` est utilisée dans une fonction de rappel afin de consigner les données de réponse brutes et les en-têtes dans le but de faciliter le débogage.

```
new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
    // Using this keyword to access AWS.Response object and properties
    console.log("Response data and headers: " + JSON.stringify(this.httpResponse));
  } else {
    console.log(data); // request succeeded
  }
});
```

En outre, étant donné que l'objet `AWS.Response` possède une propriété `Request` qui contient l'objet `AWS.Request` qui a été envoyé par l'appel de méthode d'origine, vous pouvez également accéder aux détails de la demande.

Utilisation d'un écouteur d'événements d'un objet de demande

Si vous ne créez pas de fonction de rappel anonyme et que vous n'en transmettez pas en tant que paramètre lorsque vous appelez une méthode d'objet de service, l'appel de méthode génère un objet `AWS.Request` qui doit être envoyé manuellement à l'aide de sa méthode `send`.

Pour traiter la réponse, vous devez créer un écouteur d'événements pour l'objet `AWS.Request` afin d'enregistrer une fonction de rappel pour l'appel de méthode. L'exemple suivant montre comment créer l'objet `AWS.Request` pour appeler une méthode d'objet de service et l'écouteur d'événements afin que le renvoi aboutisse.

```
// create the AWS.Request object
var request = new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances();

// register a callback event handler
request.on('success', function(response) {
  // log the successful data response
  console.log(response.data);
});

// send the request
request.send();
```

Lorsque la méthode `send` sur l'objet `AWS.Request` est appelée, le gestionnaire d'événements s'exécute lorsque l'objet de service reçoit un objet `AWS.Response`.

Pour plus d'informations sur l'`AWS.Request`objet, consultez `Class: AWS.Request`la référence de l'API. Pour plus d'informations sur l'`AWS.Response`objet, consultez Utilisation de l'objet de réponse ou consultez `Class: AWS.Response`la référence de l'API.

Création de chaînes de plusieurs rappels

Vous pouvez enregistrer plusieurs rappels sur un objet de demande. Plusieurs rappels peuvent être enregistrés pour différents événements ou pour un même événement. Vous pouvez également créer des chaînes de rappels comme l'illustré dans l'exemple suivant.

```
request.
  on('success', function(response) {
    console.log("Success!");
  }).
```

```
  on('error', function(response) {
    console.log("Error!");
  }).
  on('complete', function() {
    console.log("Always!");
  }).
  send();
```

Événements d'achèvement d'objet de demande

L'objet `AWS.Request` génère ces événements d'achèvement en fonction de la réponse de chaque méthode d'opération de service :

- `success`
- `error`
- `complete`

Vous pouvez enregistrer une fonction de rappel en réponse à ces événements. Pour obtenir la liste complète de tous les événements liés aux objets de requête, consultez `Class: AWS.Request`la référence de l'API.

L'événement success

L'événement `success` est généré en cas de réponse positive reçue de l'objet de service. Voici comment enregistrer une fonction de rappel pour cet événement.

```
request.on('success', function(response) {
  // event handler code
});
```

La réponse fournit une propriété `data` qui contient les données de la réponse sérialisée provenant du service. Par exemple, l'appel suivant à la `listBuckets` méthode de l'objet de service Amazon S3

```
s3.listBuckets.on('success', function(response) {
  console.log(response.data);
}).send();
```

renvoie la réponse, puis affiche le contenu de la propriété `data` suivant dans la console.

```
{ Owner: { ID: '...', DisplayName: '...' },
  Buckets:
   [ { Name: 'someBucketName', CreationDate: someCreationDate },
     { Name: 'otherBucketName', CreationDate: otherCreationDate } ],
  RequestId: '...' }
```

L'événement error

L'événement `error` est généré en cas de réponse d'erreur reçue de l'objet de service. Voici comment enregistrer une fonction de rappel pour cet événement.

```
request.on('error', function(error, response) {
  // event handling code
});
```

Lorsque l'événement `error` est déclenché, la valeur de la propriété `data` de la réponse est `null` et la propriété `error` contient les données relatives à l'erreur. L'objet `error` associé est transmis en tant que premier paramètre à la fonction de rappel enregistrée. Par exemple, le code suivant :

```
s3.config.credentials.accessKeyId = 'invalid';
s3.listBuckets().on('error', function(error, response) {
  console.log(error);
}).send();
```

renvoie l'erreur, puis affiche les données suivantes relatives à l'erreur dans la console.

```
{ code: 'Forbidden', message: null }
```

L'événement complete

L'événement `complete` est déclenché lorsqu'un appel d'objet de service est terminé, que l'appel ait aboutit ou non. Voici comment enregistrer une fonction de rappel pour cet événement.

```
request.on('complete', function(response) {
  // event handler code
});
```

Utilisez le rappel de l'événement `complete` pour gérer les demandes de nettoyage qui doivent s'exécuter quel que soit le résultat. Si vous utilisez des données de réponse dans un rappel

pour l'événement `complete`, commencez par vérifier les propriétés `response.data` ou `response.error` avant d'essayer d'y accéder, comme illustré dans l'exemple suivant.

```
request.on('complete', function(response) {
  if (response.error) {
    // an error occurred, handle it
  } else {
    // we can use response.data here
  }
}).send();
```

Événements HTTP d'objet de demande

L'objet `AWS.Request` génère ces événements HTTP en fonction de la réponse de chaque méthode d'opération de service :

- `httpHeaders`
- `httpData`
- `httpUploadProgress`
- `httpDownloadProgress`
- `httpError`
- `httpDone`

Vous pouvez enregistrer une fonction de rappel en réponse à ces événements. Pour obtenir la liste complète de tous les événements liés aux objets de requête, consultez `Class: AWS.Request`la référence de l'API.

L'événement httpHeaders

L'événement `httpHeaders` est déclenché lorsque des en-têtes sont envoyés par le serveur distant. Voici comment enregistrer une fonction de rappel pour cet événement.

```
request.on('httpHeaders', function(statusCode, headers, response) {
  // event handling code
});
```

Le paramètre `statusCode` pour la fonction de rappel est le code d'état HTTP. Le paramètre `headers` contient les en-têtes de la réponse.

L'événement httpData

L'événement `httpData` est déclenché pour diffuser des paquets de données de réponse à partir du service. Voici comment enregistrer une fonction de rappel pour cet événement.

```
request.on('httpData', function(chunk, response) {
  // event handling code
});
```

Cet événement est généralement utilisé pour recevoir des réponses volumineuses en plusieurs parties lorsqu'il n'est pas possible de charger l'ensemble de la réponse dans la mémoire. Cet évènement comporte un paramètre `chunk` supplémentaire qui contient une partie des données réelles provenant du serveur.

Si vous enregistrez un rappel pour l'événement `httpData`, la propriété `data` de la réponse contient la totalité de la sortie sérialisée pour la demande. Vous devez supprimer l'écouteur `httpData` par défaut si vous ne disposez pas de l'analyse et de la surcharge de mémoire supplémentaires pour les gestionnaires intégrés.

Les httpUploadProgress et httpDownloadProgress événements

L'événement `httpUploadProgress` est déclenché lorsque la demande HTTP a chargé de nouvelles données. De la même manière, l'événement `httpDownloadProgress` est déclenché lorsque la demande HTTP a téléchargé de nouvelles données. Voici comment enregistrer une fonction de rappel pour ces événements.

```
request.on('httpUploadProgress', function(progress, response) {
  // event handling code
})
.on('httpDownloadProgress', function(progress, response) {
  // event handling code
});
```

Le paramètre `progress` pour la fonction de rappel contient un objet comprenant le nombre d'octets chargés et le nombre total d'octets de la demande.

L'événement httpError

L'événement `httpError` est déclenché lorsque la demande HTTP échoue. Voici comment enregistrer une fonction de rappel pour cet événement.

```
request.on('httpError', function(error, response) {
  // event handling code
});
```

Le paramètre `error` pour la fonction de rappel contient l'erreur qui a été déclenchée.

L'événement httpDone

L'événement `httpDone` est déclenché lorsque le serveur termine l'envoi de données. Voici comment enregistrer une fonction de rappel pour cet événement.

```
request.on('httpDone', function(response) {
  // event handling code
});
```

Utilisation d'async/await

Vous pouvez utiliser le `async/await` modèle dans vos appels vers leAWS SDK for JavaScript. La plupart des fonctions qui acceptent un rappel ne renvoient pas de promesse. Comme vous n'utilisez que des `await` fonctions qui renvoient une promesse, pour utiliser le `async/await` modèle, vous devez `.promise()` enchaîner la méthode jusqu'à la fin de votre appel et supprimer le rappel.

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

```
var AWS = require("aws-sdk");
//Create an Amazon DynamoDB client service object.
dbClient = new AWS.DynamoDB({ region: "us-west-2" });
// Call DynamoDB to list existing tables
const run = async () => {
  try {
    const results = await dbClient.listTables({}).promise();
    console.log(results.TableNames.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
run();
```

> **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.

Utiliser les JavaScript promesses

La méthode `AWS.Request.promise` permet d'appeler une opération de service et de gérer le flux asynchrone au lieu d'utiliser des rappels. Dans Node.js et les scripts de navigateur, un objet `AWS.Request` est renvoyé lorsqu'une opération de service est appelée sans fonction de rappel. Vous pouvez appeler la méthode `send` de la demande pour effectuer l'appel de service.

Toutefois, `AWS.Request.promise` lance immédiatement l'appel de service et renvoie une promesse qui est soit exécutée avec la propriété `data` de la réponse, soit rejetée avec la propriété `error` de la réponse.

```
var request = new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances();

// create the promise object
var promise = request.promise();

// handle promise's fulfilled/rejected states
promise.then(
  function(data) {
    /* process the data */
  },
  function(error) {
    /* handle the error */
  }
);
```

L'exemple suivant renvoie une promesse qui est exécutée avec un objet `data` ou rejetée avec un objet `error`. Lorsque des promesses sont utilisées, un seul rappel n'est pas responsable de la détection des erreurs. Au lieu de cela, le rappel correct est appelé en fonction de la réussite ou de l'échec d'une demande.

```
var s3 = new AWS.S3({apiVersion: '2006-03-01', region: 'us-west-2'});
var params = {
```

```
  Bucket: 'bucket',
  Key: 'example2.txt',
  Body: 'Uploaded text using the promise-based method!'
};
var putObjectPromise = s3.putObject(params).promise();
putObjectPromise.then(function(data) {
  console.log('Success');
}).catch(function(err) {
  console.log(err);
});
```

Coordination de 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.

```
Promise.all([firstPromise, secondPromise, thirdPromise]).then(function(values) {

  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 the result to the caller of the Lambda function
  callback(null, values);
});
```

Prise en charge 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

Utilisation d'autres implémentations de promesses

Outre l'implémentation des promesses natives dans ECMAScript 2015, vous pouvez utiliser des bibliothèques de promesses tierces, notamment :

- bluebird
- RSVP
- Q

Ces bibliothèques de promesses facultatives peuvent s'avérer utiles si vous avez besoin que votre code s'exécute dans des environnements ne prenant pas en charge l'implémentation des promesses natives dans ECMAScript 5 et ECMAScript 2015.

Pour utiliser une bibliothèque de promesses tierce, définissez une dépendance de promesse sur le kit SDK en appelant la méthode `setPromisesDependency` de l'objet de configuration globale. Dans les scripts de navigateur, assurez-vous de charger la bibliothèque de promesses tierce avant de charger le kit SDK. Dans l'exemple suivant, le kit SDK est configuré pour utiliser l'implémentation dans la bibliothèque de promesses bluebird.

```
AWS.config.setPromisesDependency(require('bluebird'));
```

Pour revenir à l'utilisation de l'implémentation native de promesse du JavaScript moteur, appelez `setPromisesDependency` à nouveau en passant un nom de bibliothèque `null` au lieu d'un nom de bibliothèque.

Utilisation de l'objet de réponse

Une fois qu'une méthode d'objet de service a été appelée, elle renvoie un objet `AWS.Response` en le transmettant à votre fonction de rappel. Vous accédez au contenu de la réponse via les propriétés de l'objet `AWS.Response`. Pour accéder au contenu de la réponse, vous utilisez deux propriétés de l'objet `AWS.Response` :

- propriété `data`
- propriété `error`

Lorsque vous utilisez le mécanisme de rappel standard, ces deux propriétés sont fournies en tant que paramètres sur la fonction de rappel anonyme, comme illustré dans l'exemple suivant.

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

Accès aux données renvoyées dans l'objet de réponse

La propriété `data` de l'objet `AWS.Response` contient les données sérialisées renvoyées par la demande de service. Lorsque la demande aboutit, la propriété `data` contient un objet contenant un mappage pour les données renvoyées. La propriété `data` peut être null si une erreur se produit.

Voici un exemple d'appel de la `getItem` méthode d'une table DynamoDB pour récupérer le nom d'un fichier image à utiliser dans le cadre d'un jeu.

```
// Initialize parameters needed to call DynamoDB
var slotParams = {
    Key : {'slotPosition' : {N: '0'}},
    TableName : 'slotWheels',
    ProjectionExpression: 'imageFile'
};

// prepare request object for call to DynamoDB
var request = new AWS.DynamoDB({region: 'us-west-2', apiVersion:
 '2012-08-10'}).getItem(slotParams);
// log the name of the image file to load in the slot machine
request.on('success', function(response) {
    // logs a value like "cherries.jpg" returned from DynamoDB
    console.log(response.data.Item.imageFile.S);
});
// submit DynamoDB request
request.send();
```

Dans cet exemple, la table DynamoDB est une recherche d'images qui montrent les résultats d'un pull de machine à sous, tels que spécifiés par les paramètres de. `slotParams`

En cas d'appel réussi de la `getItem` méthode, la `data` propriété de l'`AWS.Response`objet contient un `Item` objet renvoyé par DynamoDB. L'accès aux données retournées dépend du paramètre `ProjectionExpression` de la demande. Dans le cas présent, il s'agit du membre `imageFile` de l'objet `Item`. Étant donné que le membre `imageFile` contient une valeur de chaîne, vous accédez au nom de fichier de l'image via la valeur du membre enfant `S` de `imageFile`.

Pagination des données retournées

Parfois, le contenu de la propriété `data` retournée par une demande de service s'étend sur plusieurs pages. Vous pouvez accéder à la page de données suivante en appelant la méthode `response.nextPage`. Cette méthode envoie une nouvelle demande. La réponse de la demande peut être capturée avec un rappel ou avec des écouteurs de réussite et d'erreur.

Vous pouvez vérifier si des pages de données supplémentaires sont disponibles pour les données renvoyées par une demande de service en appelant la méthode `response.hasNextPage`. Cette méthode renvoie une valeur booléenne pour indiquer si l'appel de la méthode `response.nextPage` renvoie des données supplémentaires.

```
s3.listObjects({Bucket: 'bucket'}).on('success', function handlePage(response) {
    // do something with response.data
    if (response.hasNextPage()) {
        response.nextPage().on('success', handlePage).send();
    }
}).send();
```

Accès aux informations sur les erreurs à partir d'un objet de réponse

La propriété `error` de l'objet `AWS.Response` contient les données d'erreur disponibles en cas d'erreur de service ou de transfert. Voici le format de l'erreur renvoyée :

```
{ code: 'SHORT_UNIQUE_ERROR_CODE', message: 'a descriptive error message' }
```

En cas d'erreur, la valeur de la propriété `data` est `null`. Si vous gérez des événements pouvant être dans un état d'échec, vérifiez toujours si la propriété `error` a été définie avant d'essayer d'accéder à la valeur de la propriété `data`.

Accès à l'objet de demande d'origine

La propriété `request` fournit un accès à l'objet `AWS.Request` d'origine. Il peut s'avérer utile pour faire référence à l'objet `AWS.Request` d'origine en vue d'accéder aux paramètres d'origine qu'il a

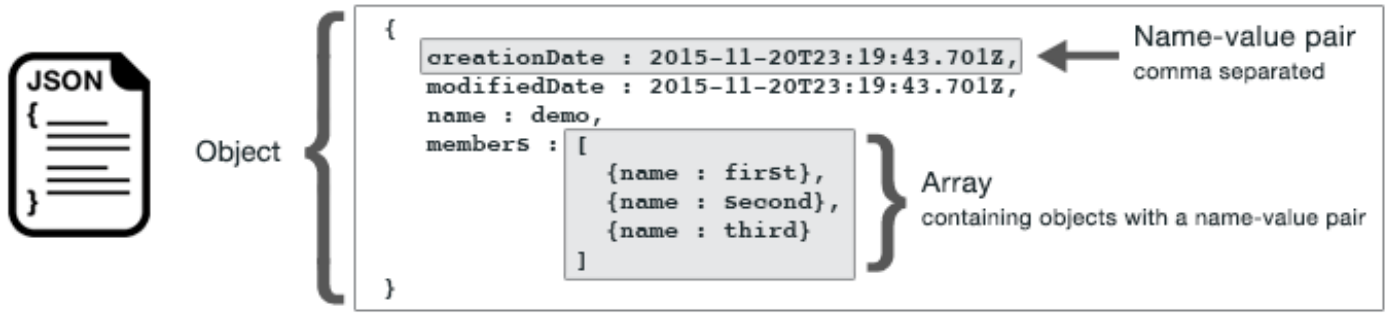
envoyés. Dans l'exemple suivant, la propriété `request` est utilisée pour accéder au paramètre `Key` de la demande de service d'origine.

```
s3.getObject({Bucket: 'bucket', Key: 'key'}).on('success', function(response) {
   console.log("Key was", response.request.params.Key);
}).send();
```

Utilisation du format JSON

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

Le SDK JavaScript utilise le JSON pour envoyer des données aux objets de service lors de 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 :

- Un objet, qui est une collection désordonnée 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.
- Un éventail, qui est un ensemble ordonné 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 des objets de service

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

```
var pullParams = {
   FunctionName : 'slotPull',
   InvocationType : 'RequestResponse',
   LogType : 'None'
};
```

L'objet `pullParams` 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`,,`InvocationType`, `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.

```
lambda = new AWS.Lambda({region: 'us-west-2', apiVersion: '2015-03-31'});
// create JSON object for service call parameters
var pullParams = {
   FunctionName : 'slotPull',
   InvocationType : 'RequestResponse',
   LogType : 'None'
};
// invoke Lambda function, passing JSON object
lambda.invoke(pullParams, function(err, data) {
   if (err) {
      console.log(err);
   } else {
```

```
      console.log(data);
   }
});
```

Renvoi de données au format JSON

Le format JSON offre un moyen standard de transmettre des données entre les différentes parties d'une application devant envoyer plusieurs valeurs en même temps. Les méthodes de classes de client dans l'API renvoient généralement des éléments JSON dans le paramètre `data` transmis à leurs fonctions de rappel. Par exemple, voici un appel à la `getBucketCors` méthode de la classe client Amazon S3.

```
// call S3 to retrieve CORS configuration for selected bucket
s3.getBucketCors(bucketParams, function(err, data) {
  if (err) {
    console.log(err);
  } else if (data) {
    console.log(JSON.stringify(data));
  }
});
```

La valeur de `data` est un objet JSON, dans cet exemple JSON qui décrit la configuration CORS actuelle pour un compartiment Amazon S3 spécifié.

```
{
   "CORSRules": [
      {
          "AllowedHeaders":["*"],
          "AllowedMethods":["POST","GET","PUT","DELETE","HEAD"],
          "AllowedOrigins":["*"],
          "ExposeHeaders":[],
          "MaxAgeSeconds":3000
      }
   ]
}
```

SDK pour les exemples de JavaScript code

Les rubriques de cette section contiennent des exemples d'utilisation du kit AWS SDK for JavaScript avec les API de différents services pour exécuter des tâches courantes.

Trouvez le code source de ces exemples et d'autres dans le référentiel d'exemples de code de AWS documentation sur GitHub. Pour proposer un nouvel exemple de code que l'équipe de documentation AWS pourrait envisager de produire, créez une nouvelle 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 relatives aux contributions.

Rubriques

CloudWatch Exemples Amazon

Amazon CloudWatch (CloudWatch) est un service Web qui surveille vos ressources Amazon Web Services et les applications que vous utilisez AWS en temps réel. Vous pouvez les utiliser CloudWatch pour collecter et suivre les métriques, qui sont des variables que vous pouvez mesurer pour vos ressources et vos applications. CloudWatch les alarmes envoient des notifications ou modifient automatiquement les ressources que vous surveillez en fonction des règles que vous définissez.

L' JavaScript API pour CloudWatch est exposée via les classes `AWS.CloudWatch`AWS.CloudWatchEvents,, et `AWS.CloudWatchLogs` client. Pour plus d'informations sur l'utilisation des classes CloudWatch clientes, consultez `Class: AWS.CloudWatch`Class: `AWS.CloudWatchEvents`, et `Class: AWS.CloudWatchLogs`dans la référence de l'API.

Rubriques

- Création d'alarmes sur Amazon CloudWatch
- Utilisation des actions d'alarme sur Amazon CloudWatch
- Obtenir des métriques auprès d'Amazon CloudWatch
- Envoyer des événements à Amazon CloudWatch Events
- Utilisation des filtres d'abonnement dans Amazon CloudWatch Logs

Création d'alarmes sur Amazon CloudWatch

Cet exemple de code Node.js présente :

- Comment récupérer des informations de base sur vos CloudWatch alarmes.
- Comment créer et supprimer une CloudWatch alarme

Scénario

Une alarme surveille une seule métrique pendant une durée que vous définissez et exécute une ou plusieurs actions en fonction de la valeur de la métrique par rapport à un seuil donné pendant un certain nombre de périodes.

Dans cet exemple, plusieurs modules Node.js sont utilisés pour créer des alarmes dans CloudWatch. Les modules Node.js utilisent le SDK pour créer des alarmes JavaScript à l'aide des méthodes suivantes de la classe `AWS.CloudWatch` client :

- `describeAlarms`
- `putMetricAlarm`
- `deleteAlarms`

Pour plus d'informations sur les CloudWatch alarmes, consultez la section Création d' CloudWatch alarmes Amazon dans le guide de CloudWatch l'utilisateur Amazon.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Description des alarmes

Créez un module Node.js nommé `cw_describealarms.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour y accéder CloudWatch, créez un objet `AWS.CloudWatch` de service. Créez un objet JSON qui contiendra les paramètres nécessaires à la récupération des descriptions d'alarmes, en limitant les alarmes renvoyées à celles dont l'état est `INSUFFICIENT_DATA`. Appelez ensuite la méthode `describeAlarms` de l'objet de service `AWS.CloudWatch`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cw_describealarms.js
```

Cet exemple de code se trouve ici sur GitHub.

Création d'une alarme pour une CloudWatch métrique

Créez un module Node.js nommé `cw_putmetricalarm.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour y accéder CloudWatch, créez un objet `AWS.CloudWatch` de service. Créez un objet JSON pour les paramètres nécessaires à la création d'une alarme basée sur une métrique, en l'occurrence l'utilisation du processeur d'une instance Amazon EC2. Les paramètres restants sont définis de façon à déclencher l'alarme lorsque la métrique dépasse un seuil de 70 %. Appelez ensuite la méthode `describeAlarms` de l'objet de service `AWS.CloudWatch`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cw_putmetricalarm.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'une alarme

Créez un module Node.js nommé `cw_deletealarms.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour y accéder CloudWatch, créez un objet `AWS.CloudWatch` de service. Créez un objet JSON contenant les noms des alarmes à supprimer. Appelez ensuite la méthode `deleteAlarms` de l'objet de service `AWS.CloudWatch`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cw_deletealarms.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des actions d'alarme sur Amazon CloudWatch

Cet exemple de code Node.js présente :

- Comment modifier automatiquement l'état de vos instances Amazon EC2 en fonction d'une CloudWatch alarme.

Scénario

À l'aide d'actions d'alarme, vous pouvez créer des alarmes qui arrêtent, mettent fin, redémarrent ou restaurent automatiquement vos instances Amazon EC2. Vous pouvez utiliser les actions d'arrêt ou de mise hors service quand vous n'avez plus besoin qu'une instance s'exécute. Vous pouvez utiliser les actions de redémarrage et de récupération pour redémarrer automatiquement ces instances.

Dans cet exemple, une série de modules Node.js sont utilisés pour définir une action d'alarme CloudWatch qui déclenche le redémarrage d'une instance Amazon EC2. Les modules Node.js utilisent le SDK pour gérer les instances Amazon EC2 JavaScript à l'aide des méthodes suivantes de `CloudWatch` la classe client :

- `enableAlarmActions`
- `disableAlarmActions`

Pour plus d'informations sur les actions CloudWatch d'alarme, consultez la section Créer des alarmes pour arrêter, résilier, redémarrer ou récupérer une instance dans le guide de CloudWatch l'utilisateur Amazon.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créez un rôle IAM dont la politique autorise la description, le redémarrage, l'arrêt ou la mise hors service d'une instance Amazon EC2. Pour plus d'informations sur la création d'un rôle IAM, consultez la section Création d'un rôle pour déléguer des autorisations à un AWS service dans le Guide de l'utilisateur IAM.

Utilisez la stratégie de rôle suivante lors de la création du rôle IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "cloudwatch:Describe*",
            "ec2:Describe*",
            "ec2:RebootInstances",
            "ec2:StopInstances*",
```

```
            "ec2:TerminateInstances"
         ],
         "Resource": [
            "*"
         ]
      }
   ]
}
```

Configurez le SDK pour JavaScript en créant un objet de configuration global, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Création et activation d'actions sur une alarme

Créez un module Node.js nommé `cw_enablealarmactions.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour y accéder CloudWatch, créez un objet `AWS.CloudWatch` de service.

Créez un objet JSON qui contiendra les paramètres nécessaires à la création d'une alarme, en spécifiant la valeur `true` pour `ActionsEnabled` et un ensemble d'ARN pour les actions qui seront déclenchées par l'alarme. Appelez la méthode `putMetricAlarm` de l'objet de service `AWS.CloudWatch`, qui crée l'alarme si elle n'existe pas ou la met à jour si elle existe.

Dans la fonction de rappel du`putMetricAlarm`, une fois l'opération terminée, créez un objet JSON contenant le nom de l' CloudWatch alarme. Appelez la méthode `enableAlarmActions` pour activer l'action d'alarme.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cw_enablealarmactions.js
```

Cet exemple de code se trouve ici sur GitHub.

Désactivation des actions sur une alarme

Créez un module Node.js nommé `cw_disablealarmactions.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour y accéder CloudWatch, créez un objet `AWS.CloudWatch` de service. Créez un objet JSON contenant le nom de l' CloudWatch alarme. Appelez la méthode `disableAlarmActions` pour désactiver les actions pour cette alarme.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cw_disablealarmactions.js
```

Cet exemple de code se trouve ici sur GitHub.

Obtenir des métriques auprès d'Amazon CloudWatch

Cet exemple de code Node.js présente :

- Comment récupérer une liste de CloudWatch métriques publiées.

- Comment publier des points de données dans CloudWatch des métriques

Scénario

Les métriques sont des données sur les performances de vos systèmes. Vous pouvez activer la surveillance détaillée de certaines ressources, telles que vos instances Amazon EC2 ou vos propres indicateurs d'application.

Dans cet exemple, une série de modules Node.js sont utilisés pour obtenir des métriques CloudWatch et envoyer des événements à Amazon CloudWatch Events. Les modules Node.js utilisent le SDK pour JavaScript obtenir des métriques CloudWatch en utilisant les méthodes suivantes de la classe `CloudWatch` client :

- `listMetrics`
- `putMetricData`

Pour plus d'informations sur CloudWatch les métriques, consultez la section Utilisation d'Amazon CloudWatch Metrics dans le guide de CloudWatch l'utilisateur Amazon.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Affichage de la liste des métriques

Créez un module Node.js nommé `cw_listmetrics.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour y accéder CloudWatch, créez un objet `AWS.CloudWatch` de service. Créez un objet JSON contenant les paramètres requis pour répertorier les métriques dans l'espace de noms `AWS/Logs`. Appelez la méthode `listMetrics` pour répertorier la métrique `IncomingLogEvents`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cw_listmetrics.js
```

Cet exemple de code se trouve ici sur GitHub.

Envoi de métriques personnalisées

Créez un module Node.js nommé `cw_putmetricdata.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour y accéder CloudWatch, créez un objet `AWS.CloudWatch` de service. Créez un objet JSON contenant les paramètres requis afin d'envoyer un point de données pour la métrique personnalisée `PAGES_VISITED`. Appelez la méthode `putMetricData`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cw_putmetricdata.js
```

Cet exemple de code se trouve ici sur GitHub.

Envoyer des événements à Amazon CloudWatch Events

Cet exemple de code Node.js présente :

- Procédure de création et de mise à jour d'une règle utilisée pour déclencher un événement.
- Procédure de définition d'une ou plusieurs cibles pour répondre à un événement.
- Procédure d'envoi des événements associés à des cibles pour traitement.

Scénario

CloudWatch Events fournit un flux en temps quasi réel d'événements système décrivant les modifications apportées aux ressources Amazon Web Services à différentes cibles. À l'aide de règles simples, vous pouvez faire correspondre les événements et les acheminer vers un ou plusieurs flux ou fonctions cibles.

Dans cet exemple, une série de modules Node.js sont utilisés pour envoyer des CloudWatch événements à Events. Les modules Node.js utilisent le SDK pour gérer les instances JavaScript à l'aide des méthodes suivantes de la classe `CloudWatchEvents` client :

- `putRule`
- `putTargets`
- `putEvents`

Pour plus d'informations sur les CloudWatch événements, consultez la section Ajouter des événements PutEvents dans le guide de l'utilisateur Amazon CloudWatch Events.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créez une fonction Lambda à l'aide du plan hello-world pour servir de cible aux événements. Pour savoir comment procéder, consultez Étape 1 : création d'une AWS Lambda fonction dans le guide de l'utilisateur Amazon CloudWatch Events.

- Créez un rôle IAM dont la politique accorde l'autorisation aux CloudWatch événements, y compris `events.amazonaws.com` en tant qu'entité de confiance. Pour plus d'informations sur la création d'un rôle IAM, consultez la section Création d'un rôle pour déléguer des autorisations à un AWS service dans le Guide de l'utilisateur IAM.

Utilisez la stratégie de rôle suivante lors de la création du rôle IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "CloudWatchEventsFullAccess",
         "Effect": "Allow",
         "Action": "events:*",
         "Resource": "*"
      },
      {
         "Sid": "IAMPassRoleForCloudWatchEvents",
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "arn:aws:iam::*:role/AWS_Events_Invoke_Targets"
      }
   ]
}
```

Utilisez la relation d'approbation suivante lors de la création du rôle IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "events.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

Création d'une règle planifiée

Créez un module Node.js nommé `cwe_putrule.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder aux CloudWatch événements, créez un objet `AWS.CloudWatchEvents` de service. Créez un objet JSON contenant les paramètres requis pour spécifier la nouvelle règle planifiée, avec les éléments suivants :

- Nom de la règle
- L'ARN du rôle IAM que vous avez créé précédemment
- Une expression pour planifier le déclenchement de la règle toutes les cinq minutes

Appelez la méthode `putRule` pour créer la règle. Le rappel renvoie l'ARN de la nouvelle règle ou de la règle modifiée.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cwe_putrule.js
```

Cet exemple de code se trouve ici sur GitHub.

Ajouter une cible AWS Lambda de fonction

Créez un module Node.js nommé `cwe_puttargets.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder aux CloudWatch événements, créez un objet `AWS.CloudWatchEvents` de service. Créez un objet JSON contenant les paramètres nécessaires pour spécifier la règle à laquelle vous souhaitez associer la cible, y compris l'ARN de la fonction Lambda que vous avez créée. Appelez la méthode `putTargets` de l'objet de service `AWS.CloudWatchEvents`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cwe_puttargets.js
```

Cet exemple de code se trouve ici sur GitHub.

Envoi d'événements

Créez un module Node.js nommé `cwe_putevents.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder aux CloudWatch événements, créez un objet `AWS.CloudWatchEvents` de service. Créez un objet JSON contenant les paramètres requis pour envoyer des événements. Pour chaque événement, incluez la source de l'événement, les ARN des ressources affectées par l'événement et les détails de l'événement. Appelez la méthode `putEvents` de l'objet de service `AWS.CloudWatchEvents`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cwe_putevents.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des filtres d'abonnement dans Amazon CloudWatch Logs

Cet exemple de code Node.js présente :

- Comment créer et supprimer des filtres pour les événements de journalisation dans CloudWatch Logs

Scénario

Les abonnements permettent d'accéder à un flux en temps réel des événements du journal depuis CloudWatch Logs et de transmettre ce flux à d'autres services, tels qu'un flux Amazon Kinesis ou à des fins de traitementAWS Lambda, d'analyse ou de chargement personnalisés sur d'autres systèmes. Un filtre d'abonnement définit le modèle à utiliser pour filtrer les événements du journal transmis à votre AWS ressource.

Dans cet exemple, une série de modules Node.js sont utilisés pour répertorier, créer et supprimer un filtre d'abonnement dans CloudWatch Logs. La destination des événements du journal est une fonction Lambda. Les modules Node.js utilisent le SDK pour gérer les filtres d'abonnement JavaScript à l'aide des méthodes suivantes de la classe `CloudWatchLogs` client :

- `putSubscriptionFilters`
- `describeSubscriptionFilters`
- `deleteSubscriptionFilter`

Pour plus d'informations sur CloudWatch les abonnements aux journaux, consultez la section Traitement en temps réel des données de journal avec les abonnements dans le guide de l'utilisateur Amazon CloudWatch Logs.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.

- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créez une fonction Lambda comme destination pour les événements du journal. Vous devrez utiliser l'ARN de cette fonction. Pour plus d'informations sur la configuration d'une fonction Lambda, consultez la section Filtres d'abonnement AWS Lambda dans le guide de l'utilisateur Amazon CloudWatch Logs.
- Créez un rôle IAM dont la politique autorise l'appel de la fonction Lambda que vous avez créée et accorde un accès complet CloudWatch aux journaux ou appliquez la politique suivante au rôle d'exécution que vous créez pour la fonction Lambda. Pour plus d'informations sur la création d'un rôle IAM, consultez la section Création d'un rôle pour déléguer des autorisations à un AWS service dans le Guide de l'utilisateur IAM.

Utilisez la stratégie de rôle suivante lors de la création du rôle IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "lambda:InvokeFunction"
         ],
         "Resource": [
            "*"
         ]
      }
   ]
}
```

Description des filtres d'abonnements existants

Créez un module Node.js nommé `cwl_describesubscriptionfilters.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder aux CloudWatch journaux, créez un objet `AWS.CloudWatchLogs` de service. Créez un objet JSON contenant les paramètres requis pour décrire vos filtres existants, y compris le nom du groupe de journaux et le nombre maximal de filtres à décrire. Appelez la méthode `describeSubscriptionFilters`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cwl_describesubscriptionfilters.js
```

Cet exemple de code se trouve ici sur GitHub.

Création d'un filtre d'abonnement

Créez un module Node.js nommé `cwl_putsubscriptionfilter.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder aux CloudWatch journaux, créez un objet `AWS.CloudWatchLogs` de service. Créez un objet JSON contenant les paramètres nécessaires à la création d'un filtre, notamment l'ARN de la fonction Lambda de destination, le nom du filtre,

le modèle de chaîne pour le filtrage et le nom du groupe de journaux. Appelez la méthode `putSubscriptionFilters`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cwl_putsubscriptionfilter.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un filtre d'abonnement

Créez un module Node.js nommé `cwl_deletesubscriptionfilters.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder aux CloudWatch journaux, créez un objet `AWS.CloudWatchLogs` de service. Créez un objet JSON contenant les paramètres requis pour supprimer un filtre, notamment les noms du filtre et du groupe de journaux. Appelez la méthode `deleteSubscriptionFilters`.

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

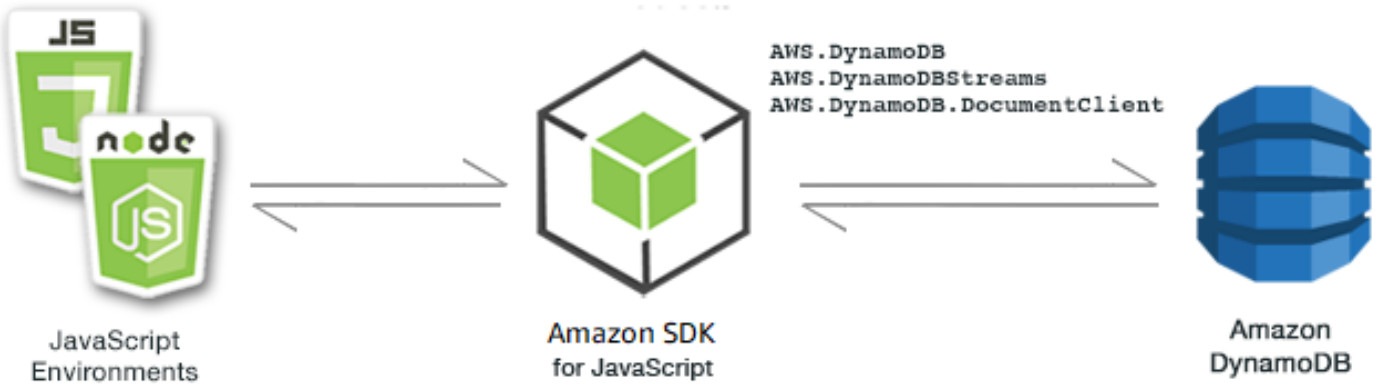
Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node cwl_deletesubscriptionfilter.js
```

Cet exemple de code se trouve ici sur GitHub.

Exemples 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 `AWS.DynamoDB` des classes`AWS.DynamoDBStreams`,, `AWS.DynamoDB.DocumentClient` et client. Pour plus d'informations sur l'utilisation des classes clientes DynamoDB, `Class: AWS.DynamoDB`consultez `Class: AWS.DynamoDBStreams`, `Class: AWS.DynamoDB.DocumentClient`et dans la référence de l'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.

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 `AWS.DynamoDB` client :

- createTable
- listTables

- describeTable
- deleteTable

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Création d'une table

Créez un module Node.js nommé `ddb_createtable.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `createTable` méthode de l'objet de service DynamoDB.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_createtable.js
```

Cet exemple de code se trouve ici sur GitHub.

Liste de vos tables

Créez un module Node.js nommé `ddb_listtables.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `listTables` méthode de l'objet de service DynamoDB.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_listtables.js
```

Cet exemple de code se trouve ici sur GitHub.

Description d'une table

Créez un module Node.js nommé `ddb_describetable.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `describeTable` méthode de l'objet de service DynamoDB.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_describetable.js TABLE_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'une table

Créez un module Node.js nommé `ddb_deletetable.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `deleteTable` méthode de l'objet de service DynamoDB.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_deletetable.js TABLE_NAME
```

Cet exemple de code se trouve ici sur 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

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 `AWS.DynamoDB` classe client :

- putItem
- getItem
- deleteItem

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.

- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- 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

Ecriture d'un élément

Créez un module Node.js nommé `ddb_putitem.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `putItem` méthode de l'objet de service DynamoDB.

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

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_putitem.js
```

Cet exemple de code se trouve ici sur GitHub.

Obtention d'un élément

Créez un module Node.js nommé `ddb_getitem.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `getItem` 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 `getItem` méthode de l'objet de service DynamoDB.

```
// 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);
  }
```

```
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_getitem.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un élément

Créez un module Node.js nommé `ddb_deleteitem.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `deleteItem` méthode de l'objet de service DynamoDB.

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

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_deleteitem.js
```

Cet exemple de code se trouve ici sur 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

Scénario

Dans cet exemple, vous utilisez une série de modules Node.js pour placer un lot d'éléments dans une table DynamoDB ainsi que pour 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 :

- batchGetItem
- batchWriteItem

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- 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

Lecture des éléments par lots

Créez un module Node.js nommé `ddb_batchgetitem.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `batchGetItem` méthode de l'objet de service DynamoDB.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_batchgetitem.js
```

Cet exemple de code se trouve ici sur GitHub.

Écriture d'éléments par lots

Créez un module Node.js nommé `ddb_batchwriteitem.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. Créez un objet JSON contenant les paramètres requis pour obtenir un lot d'éléments, ce qui dans cet exemple inclut la table dans laquelle vous souhaitez écrire des éléments, la ou les clés à écrire pour chaque élément, ainsi que les attributs et leurs valeurs. Appelez la `batchWriteItem` méthode de l'objet de service DynamoDB.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_batchwriteitem.js
```

Cet exemple de code se trouve ici sur 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.

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 :

- query
- scan

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- 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

Interrogation d'une table

Cet exemple interroge une table qui contient des informations sur chaque épisode d'une série de vidéos, en renvoyant les titres et les sous-titres des épisodes de la deuxième saison après l'épisode 9 qui contiennent une expression spécifiée dans leur sous-titre.

Créez un module Node.js nommé `ddb_query.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `query` méthode de l'objet de service DynamoDB.

```
// 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 = {
  ExpressionAttributeValues: {
```

```
    ":s": { N: "2" },
    ":e": { N: "09" },
    ":topic": { S: "PHRASE" },
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  ProjectionExpression: "Episode, Title, Subtitle",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

ddb.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    //console.log("Success", data.Items);
    data.Items.forEach(function (element, index, array) {
      console.log(element.Title.S + " (" + element.Subtitle.S + ")");
    });
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_query.js
```

Cet exemple de code se trouve ici sur GitHub.

Analyse d'une table

Créez un module Node.js nommé `ddb_scan.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un `AWS.DynamoDB` objet de service. 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 `scan` méthode de l'objet de service DynamoDB.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddb_scan.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation du client de documents DynamoDB

Cet exemple de code Node.js présente :

- Comment accéder à une table DynamoDB à l'aide du client de documents

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 la classe DynamoDB Document Client, `AWS.DynamoDB.DocumentClient`consultez le Guide de référence des API. 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 du client de documents. Le code utilise le SDK pour interroger et scanner des tables JavaScript à l'aide des méthodes suivantes de la classe DynamoDB Document Client :

- get
- put
- update
- query
- supprimer

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.

- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- 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.

Obtention d'un élément à partir d'une table

Créez un module Node.js nommé `ddbdoc_get.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un objet. `AWS.DynamoDB.DocumentClient` Créez un objet JSON contenant les paramètres requis pour obtenir un élément de la table, ce qui dans cet exemple inclut le nom de la table, le nom de la clé de hachage dans cette table et la valeur de la clé de hachage de l'élément que vous récupérez. Appelez la `get` méthode du client de documents 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddbdoc_get.js
```

Cet exemple de code se trouve ici sur GitHub.

Positionnement d'un élément dans une table

Créez un module Node.js nommé `ddbdoc_put.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un objet. `AWS.DynamoDB.DocumentClient` Créez un objet JSON contenant les paramètres requis pour écrire un élément dans la table, ce 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 `put` méthode du client de documents 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddbdoc_put.js
```

Cet exemple de code se trouve ici sur GitHub.

Mise à jour d'un élément dans une table

Créez un module Node.js nommé `ddbdoc_update.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un objet. `AWS.DynamoDB.DocumentClient` Créez un objet JSON contenant les paramètres requis pour écrire un élément dans la table, ce qui dans cet exemple inclut le nom de la table, la clé de l'élément à mettre à jour et un ensemble de `UpdateExpressions` définissant les attributs de l'élément à mettre à jour avec les jetons auxquels vous attribuez des valeurs dans les paramètres `ExpressionAttributeValues`. Appelez la `update` méthode du client de documents 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" });

// Create variables to hold numeric key values
var season = SEASON_NUMBER;
var episode = EPISODES_NUMBER;

var params = {
  TableName: "EPISODES_TABLE",
  Key: {
    Season: season,
    Episode: episode,
  },
  UpdateExpression: "set Title = :t, Subtitle = :s",
  ExpressionAttributeValues: {
    ":t": "NEW_TITLE",
    ":s": "NEW_SUBTITLE",
  },
};

docClient.update(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
```

```
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddbdoc_update.js
```

Cet exemple de code se trouve ici sur GitHub.

Interrogation d'une table

Cet exemple interroge une table qui contient des informations sur chaque épisode d'une série de vidéos, en renvoyant les titres et les sous-titres des épisodes de la deuxième saison après l'épisode 9 qui contiennent une expression spécifiée dans leur sous-titre.

Créez un module Node.js nommé `ddbdoc_query.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un objet. `AWS.DynamoDB.DocumentClient` 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 `query` méthode du client de documents 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 = {
  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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddbdoc_query.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un élément d'une table

Créez un module Node.js nommé `ddbdoc_delete.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à DynamoDB, créez un objet. `AWS.DynamoDB.DocumentClient` Créez un objet JSON contenant les paramètres requis pour supprimer un élément de la table, ce 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 à supprimer. Appelez la `delete` méthode du client de documents 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ddbdoc_delete.js
```

Cet exemple de code se trouve ici sur GitHub.

Exemples Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) est un service Web qui fournit un hébergement de serveurs virtuels dans le cloud. Destiné aux développeurs, il est conçu pour faciliter les opérations de cloud computing à l'échelle du web en offrant une capacité de calcul redimensionnable.

L' JavaScript API pour Amazon EC2 est exposée par le biais de la classe `AWS.EC2` client. Pour plus d'informations sur l'utilisation de la classe client Amazon EC2, consultez la référence `Class: AWS.EC2`de l'API.

Rubriques

- Création d'une instance Amazon EC2
- Gestion des instances Amazon EC2
- Utilisation des paires de clés Amazon EC2
- Utilisation des régions et des zones de disponibilité avec Amazon EC2
- Utilisation des groupes de sécurité dans Amazon EC2
- Utilisation d'adresses IP élastiques dans Amazon EC2

Création d'une instance Amazon EC2

Cet exemple de code Node.js présente :

- Comment créer une instance Amazon EC2 à partir d'une Amazon Machine Image (AMI) publique.
- Comment créer et attribuer des balises à la nouvelle instance Amazon EC2.

À propos de l'exemple

Dans cet exemple, vous utilisez un module Node.js pour créer une instance Amazon EC2 et lui attribuer à la fois une paire de clés et des balises. Le code utilise le SDK pour JavaScript créer et étiqueter une instance en utilisant les méthodes suivantes de la classe client Amazon EC2 :

- `runInstances`
- `createTags`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créer une paire de clé. Pour plus de détails, consultez Utilisation des paires de clés Amazon EC2. Vous utilisez le nom de la paire de clés dans cet exemple.

Création et balisage d'une instance

Créez un module Node.js nommé `ec2_createinstances.js`. Veillez à configurer le kit SDK comme indiqué précédemment.

Créez un objet qui permettra de transmettre les paramètres pour la méthode `runInstances` de la classe client AWS.EC2, y compris le nom de la paire de clés à attribuer et l'ID de l'AMI à exécuter. Pour appeler la méthode `runInstances`, créez une promesse pour appeler un objet de service Amazon EC2, en transmettant les paramètres. Traitez ensuite l'élément response dans le rappel de promesse.

Le code ajoute ensuite une `Name` balise à une nouvelle instance, que la console Amazon EC2 reconnaît et affiche dans le champ Nom de la liste des instances. Vous pouvez ajouter jusqu'à 50 balises à une instance, qui peuvent toutes être ajoutées dans un seul appel à la méthode `createTags`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Load credentials and set region from JSON file
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// AMI is amzn-ami-2011.09.1.x86_64-ebs
var instanceParams = {
  ImageId: "AMI_ID",
  InstanceType: "t2.micro",
  KeyName: "KEY_PAIR_NAME",
  MinCount: 1,
  MaxCount: 1,
};

// Create a promise on an EC2 service object
var instancePromise = new AWS.EC2({ apiVersion: "2016-11-15" })
  .runInstances(instanceParams)
  .promise();

// Handle promise's fulfilled/rejected states
instancePromise
  .then(function (data) {
    console.log(data);
    var instanceId = data.Instances[0].InstanceId;
    console.log("Created instance", instanceId);
    // Add tags to the instance
    tagParams = {
      Resources: [instanceId],
```

```
      Tags: [
        {
          Key: "Name",
          Value: "SDK Sample",
        },
      ],
    };
    // Create a promise on an EC2 service object
    var tagPromise = new AWS.EC2({ apiVersion: "2016-11-15" })
      .createTags(tagParams)
      .promise();
    // Handle promise's fulfilled/rejected states
    tagPromise
      .then(function (data) {
        console.log("Instance tagged");
      })
      .catch(function (err) {
        console.error(err, err.stack);
      });
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_createinstances.js
```

Cet exemple de code se trouve ici sur GitHub.

Gestion des instances Amazon EC2

Cet exemple de code Node.js présente :

- Comment récupérer des informations de base sur vos instances Amazon EC2.
- Comment démarrer et arrêter la surveillance détaillée d'une instance Amazon EC2
- Comment démarrer et arrêter une instance Amazon EC2

- Comment redémarrer une instance Amazon EC2

Scénario

Dans cet exemple, vous utilisez plusieurs modules Node.js pour effectuer des opérations de gestion des instances de base. Les modules Node.js utilisent le SDK JavaScript pour gérer les instances en utilisant les méthodes de classe client Amazon EC2 suivantes :

- `describeInstances`
- `monitorInstances`
- `unmonitorInstances`
- `startInstances`
- `stopInstances`
- `rebootInstances`

Pour plus d'informations sur le cycle de vie des instances Amazon EC2, consultez la section relative au cycle de vie des instances dans le guide de l'utilisateur Amazon EC2.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créez une instance Amazon EC2. Pour plus d'informations sur la création d'instances Amazon EC2, consultez les instances Amazon EC2 dans le guide de l'utilisateur Amazon EC2 ou les instances Amazon EC2 dans le guide de l'utilisateur Amazon EC2.

Description des instances

Créez un module Node.js nommé `ec2_describeinstances.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service.

Appelez la `describeInstances` méthode de l'objet de service Amazon EC2 pour obtenir une description détaillée de vos instances.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  DryRun: false,
};

// Call EC2 to retrieve policy for selected bucket
ec2.describeInstances(params, function (err, data) {
  if (err) {
    console.log("Error", err.stack);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_describeinstances.js
```

Cet exemple de code se trouve ici sur GitHub.

Gestion de la surveillance des instances

Créez un module Node.js nommé `ec2_monitorinstances.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Ajoutez les ID des instances pour lesquelles vous souhaitez contrôler la surveillance.

Sur la base de la valeur d'un argument de ligne de commande (`0Nou0FF`), appelez soit la `monitorInstances` méthode de l'objet de service Amazon EC2 pour commencer la surveillance détaillée des instances spécifiées, soit appelez la méthode. `unmonitorInstances` Utilisez le paramètre `DryRun` pour vérifier si vous pouvez modifier la surveillance des instances avant de le faire réellement.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  InstanceIds: ["INSTANCE_ID"],
  DryRun: true,
};

if (process.argv[2].toUpperCase() === "ON") {
  // Call EC2 to start monitoring the selected instances
  ec2.monitorInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.monitorInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.InstanceMonitorings);
        }
      });
    } else {
      console.log("You don't have permission to change instance monitoring.");
    }
  });
} else if (process.argv[2].toUpperCase() === "OFF") {
  // Call EC2 to stop monitoring the selected instances
  ec2.unmonitorInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.unmonitorInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.InstanceMonitorings);
        }
      });
    } else {
      console.log("You don't have permission to change instance monitoring.");
```

```
    }
  });
}
```

Pour exécuter l'exemple, entrez ce qui suit sur la ligne de commande, en spécifiant `ON` pour commencer la surveillance détaillée ou `OFF` pour l'interrompre.

```
node ec2_monitorinstances.js ON
```

Cet exemple de code se trouve ici sur GitHub.

Démarrage et arrêt des instances

Créez un module Node.js nommé `ec2_startstopinstances.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Ajoutez les ID des instances que vous souhaitez démarrer ou arrêter.

Sur la base de la valeur d'un argument de ligne de commande (`STARTouSTOP`), appelez soit la `startInstances` méthode de l'objet de service Amazon EC2 pour démarrer les instances spécifiées, soit la méthode pour `stopInstances` les arrêter. Utilisez le paramètre `DryRun` pour vérifier si vous y êtes autorisé avant de tenter de démarrer ou d'arrêter les instances sélectionnées.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  InstanceIds: [process.argv[3]],
  DryRun: true,
};

if (process.argv[2].toUpperCase() === "START") {
  // Call EC2 to start the selected instances
  ec2.startInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.startInstances(params, function (err, data) {
        if (err) {
```

```
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.StartingInstances);
        }
      });
    } else {
      console.log("You don't have permission to start instances.");
    }
  });
} else if (process.argv[2].toUpperCase() === "STOP") {
  // Call EC2 to stop the selected instances
  ec2.stopInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.stopInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.StoppingInstances);
        }
      });
    } else {
      console.log("You don't have permission to stop instances");
    }
  });
}
```

Pour exécuter l'exemple, entrez ce qui suit sur la ligne de commande, en spécifiant `START` pour démarrer les instances ou `STOP` pour les arrêter.

```
node ec2_startstopinstances.js START INSTANCE_ID
```

Cet exemple de code se trouve ici sur GitHub.

Redémarrage des instances

Créez un module Node.js nommé `ec2_rebootinstances.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de service Amazon EC2. Ajoutez les ID des instances que vous souhaitez redémarrer. Appelez la méthode `rebootInstances` de l'objet de service `AWS.EC2` pour redémarrer les instances spécifiées. Utilisez le paramètre `DryRun` pour savoir si vous y êtes autorisé avant de tenter de redémarrer ces instances.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  InstanceIds: ["INSTANCE_ID"],
  DryRun: true,
};

// Call EC2 to reboot instances
ec2.rebootInstances(params, function (err, data) {
  if (err && err.code === "DryRunOperation") {
    params.DryRun = false;
    ec2.rebootInstances(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else if (data) {
        console.log("Success", data);
      }
    });
  } else {
    console.log("You don't have permission to reboot instances.");
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_rebootinstances.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des paires de clés Amazon EC2

Cet exemple de code Node.js présente :

- Procédure de récupération des informations sur les paires de clés
- Comment créer une paire de clés pour accéder à une instance Amazon EC2
- Procédure de suppression d'une paire de clés existante.

Scénario

Amazon EC2 utilise le chiffrement à clé publique pour chiffrer et déchiffrer les informations de connexion. Le chiffrement à clé publique a recours à une clé publique pour chiffrer les données ; le destinataire utilise ensuite la clé privée pour déchiffrer les données. La clé publique et la clé privée constituent une paire de clés.

Dans cet exemple, vous utilisez une série de modules Node.js pour effectuer plusieurs opérations de gestion des paires de clés Amazon EC2. Les modules Node.js utilisent le SDK JavaScript pour gérer les instances en utilisant les méthodes suivantes de la classe client Amazon EC2 :

- `createKeyPair`
- `deleteKeyPair`
- `describeKeyPairs`

Pour plus d'informations sur les paires de clés Amazon EC2, consultez les paires de clés Amazon EC2 dans le guide de l'utilisateur Amazon EC2 ou les paires de clés Amazon EC2 et les instances Windows dans le guide de l'utilisateur Amazon EC2.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Description de vos paires de clés

Créez un module Node.js nommé `ec2_describekeypairs.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez un objet JSON vide qui contiendra les paramètres requis par la méthode `describeKeyPairs` pour renvoyer les descriptions de toutes les paires de clés. Vous pouvez également fournir une liste des noms de paires de clés dans la portion `KeyName` des paramètres du fichier JSON à la méthode `describeKeyPairs`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// Retrieve key pair descriptions; no params needed
ec2.describeKeyPairs(function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.KeyPairs));
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_describekeypairs.js
```

Cet exemple de code se trouve ici sur GitHub.

Création d'une paire de clés

Chaque paire de clés doit avoir un nom. Amazon EC2 associe la clé publique au nom de clé que vous spécifiez. Créez un module Node.js nommé `ec2_createkeypair.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez les paramètres JSON afin de spécifier le nom de la paire de clés, puis transmettez-les pour appeler la méthode `createKeyPair`.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  KeyName: "KEY_PAIR_NAME",
};

// Create the key pair
ec2.createKeyPair(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log(JSON.stringify(data));
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_createkeypair.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'une paire de clés

Créez un module Node.js nommé `ec2_deletekeypair.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez les paramètres JSON pour spécifier le nom de la paire de clés à supprimer. Ensuite, appelez la méthode `deleteKeyPair`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
```

```
  KeyName: "KEY_PAIR_NAME",
};

// Delete the key pair
ec2.deleteKeyPair(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Key Pair Deleted");
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_deletekeypair.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des régions et des zones de disponibilité avec Amazon EC2

Cet exemple de code Node.js présente :

- Procédure de récupération des descriptions pour les régions et les zones de disponibilité.

Scénario

Amazon EC2 est hébergé sur plusieurs sites dans le monde entier. Ces emplacements sont composés de régions et de zones de disponibilité. Chaque région constitue une zone géographique séparée. Chaque région se compose de plusieurs emplacements isolés appelés zones de disponibilité. Amazon EC2 permet de placer des instances et des données sur plusieurs sites.

Dans cet exemple, vous utilisez plusieurs modules Node.js pour récupérer des détails sur les régions et les zones de disponibilité. Les modules Node.js utilisent le SDK JavaScript pour gérer les instances en utilisant les méthodes suivantes de la classe client Amazon EC2 :

- `describeAvailabilityZones`

- `describeRegions`

Pour plus d'informations sur les régions et les zones de disponibilité, consultez Régions et zones de disponibilité dans le guide de l'utilisateur Amazon EC2 ou Régions et zones de disponibilité dans le guide de l'utilisateur Amazon EC2.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Description des régions et zones de disponibilité

Créez un module Node.js nommé `ec2_describeregionsandzones.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez un objet JSON vide à transmettre en tant que paramètres, qui renvoie toutes les descriptions disponibles. Appelez ensuite les méthodes `describeRegions` et `describeAvailabilityZones`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {};

// Retrieves all regions/endpoints that work with EC2
ec2.describeRegions(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Regions: ", data.Regions);
  }
});

// Retrieves availability zones only for region of the ec2 service object
ec2.describeAvailabilityZones(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Availability Zones: ", data.AvailabilityZones);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_describeregionsandzones.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des groupes de sécurité dans Amazon EC2

Cet exemple de code Node.js présente :

- Procédure de récupération des informations sur vos groupes de sécurité.
- Comment créer un groupe de sécurité pour accéder à une instance Amazon EC2
- Procédure de suppression d'un groupe de sécurité existant.

Scénario

Un groupe de sécurité Amazon EC2 agit comme un pare-feu virtuel qui contrôle le trafic d'une ou de plusieurs instances. Vous ajoutez des règles à chaque groupe de sécurité pour autoriser le trafic vers ou depuis ses instances associées. Vous pouvez modifier les règles d'un groupe de sécurité à tout moment. Les nouvelles règles sont appliquées automatiquement à toutes les instances associées au groupe de sécurité.

Dans cet exemple, vous utilisez une série de modules Node.js pour effectuer plusieurs opérations Amazon EC2 impliquant des groupes de sécurité. Les modules Node.js utilisent le SDK JavaScript pour gérer les instances en utilisant les méthodes suivantes de la classe client Amazon EC2 :

- `describeSecurityGroups`
- `authorizeSecurityGroupIngress`
- `createSecurityGroup`
- `describeVpcs`
- `deleteSecurityGroup`

Pour plus d'informations sur les groupes de sécurité Amazon EC2, consultez Amazon EC2 Amazon Security Groups for Linux Instances dans le Guide de l'utilisateur Amazon EC2 ou Amazon EC2 Security Groups for Windows Instances dans le Guide de l'utilisateur Amazon EC2.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Description de vos groupes de sécurité

Créez un module Node.js nommé `ec2_describesecuritygroups.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez un objet JSON à transmettre en tant que paramètres, en incluant les ID des groupes de sécurité à décrire. Appelez ensuite la `describeSecurityGroups` méthode de l'objet de service Amazon EC2.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
```

```
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  GroupIds: ["SECURITY_GROUP_ID"],
};

// Retrieve security group descriptions
ec2.describeSecurityGroups(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.SecurityGroups));
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_describesecuritygroups.js
```

Cet exemple de code se trouve ici sur GitHub.

Création d'un groupe et de règles de sécurité

Créez un module Node.js nommé `ec2_createsecuritygroup.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez un objet JSON pour les paramètres qui spécifient le nom du groupe de sécurité, une description et l'ID du VPC. Transmettez les paramètres à la méthode `createSecurityGroup`.

Après avoir créé le groupe de sécurité, vous pouvez définir des règles autorisant le trafic entrant. Créez un objet JSON pour les paramètres qui spécifient le protocole IP et les ports entrants sur lesquels l'instance Amazon EC2 recevra le trafic. Transmettez les paramètres à la méthode `authorizeSecurityGroupIngress`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Load credentials and set region from JSON file
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// Variable to hold a ID of a VPC
```

```
var vpc = null;

// Retrieve the ID of a VPC
ec2.describeVpcs(function (err, data) {
  if (err) {
    console.log("Cannot retrieve a VPC", err);
  } else {
    vpc = data.Vpcs[0].VpcId;
    var paramsSecurityGroup = {
      Description: "DESCRIPTION",
      GroupName: "SECURITY_GROUP_NAME",
      VpcId: vpc,
    };
    // Create the instance
    ec2.createSecurityGroup(paramsSecurityGroup, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        var SecurityGroupId = data.GroupId;
        console.log("Success", SecurityGroupId);
        var paramsIngress = {
          GroupId: "SECURITY_GROUP_ID",
          IpPermissions: [
            {
              IpProtocol: "tcp",
              FromPort: 80,
              ToPort: 80,
              IpRanges: [{ CidrIp: "0.0.0.0/0" }],
            },
            {
              IpProtocol: "tcp",
              FromPort: 22,
              ToPort: 22,
              IpRanges: [{ CidrIp: "0.0.0.0/0" }],
            },
          ],
        };
        ec2.authorizeSecurityGroupIngress(paramsIngress, function (err, data) {
          if (err) {
            console.log("Error", err);
          } else {
            console.log("Ingress Successfully Set", data);
          }
        });
```

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

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_createsecuritygroup.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un groupe de sécurité

Créez un module Node.js nommé `ec2_deletesecuritygroup.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez les paramètres JSON pour spécifier le nom du groupe de sécurité à supprimer. Ensuite, appelez la méthode `deleteSecurityGroup`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  GroupId: "SECURITY_GROUP_ID",
};

// Delete the security group
ec2.deleteSecurityGroup(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Security Group Deleted");
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_deletesecuritygroup.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation d'adresses IP élastiques dans Amazon EC2

Cet exemple de code Node.js présente :

- Procédure de récupération des descriptions de vos adresses IP Elastic.
- Procédure d'attribution et de libération d'une adresse IP Elastic.
- Comment associer une adresse IP élastique à une instance Amazon EC2

Scénario

Une adresse IP Elastic est une adresse IP statique conçue pour le cloud computing dynamique. Une adresse IP élastique est associée à votre AWS compte. Il s'agit d'une adresse IP publique accessible depuis Internet. Si votre instance ne dispose pas d'une adresse IP publique, vous pouvez associer une adresse IP Elastic à votre instance pour établir la communication avec Internet.

Dans cet exemple, vous utilisez une série de modules Node.js pour effectuer plusieurs opérations Amazon EC2 impliquant des adresses IP Elastic. Les modules Node.js utilisent le SDK JavaScript pour gérer les adresses IP élastiques en utilisant les méthodes suivantes de la classe client Amazon EC2 :

- `describeAddresses`
- `allocateAddress`
- `associateAddress`
- `releaseAddress`

Pour plus d'informations sur les adresses IP élastiques dans Amazon EC2, consultez Adresses IP élastiques dans le guide de l'utilisateur Amazon EC2 ou Adresses IP élastiques dans le guide de l'utilisateur Amazon EC2.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créez une instance Amazon EC2. Pour plus d'informations sur la création d'instances Amazon EC2, consultez les instances Amazon EC2 dans le guide de l'utilisateur Amazon EC2 ou les instances Amazon EC2 dans le guide de l'utilisateur Amazon EC2.

Description des adresses IP Elastic

Créez un module Node.js nommé `ec2_describeaddresses.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez un objet JSON à transmettre en tant que paramètres, en filtrant les adresses renvoyées selon celles de votre VPC. Pour récupérer les descriptions de toutes vos adresses IP Elastic, il suffit d'omettre un filtre de l'objet JSON des paramètres. Appelez ensuite la `describeAddresses` méthode de l'objet de service Amazon EC2.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  Filters: [{ Name: "domain", Values: ["vpc"] }],
};

// Retrieve Elastic IP address descriptions
ec2.describeAddresses(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.Addresses));
```

```
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_describeaddresses.js
```

Cet exemple de code se trouve ici sur GitHub.

Allocation et association d'une adresse IP élastique à une instance Amazon EC2

Créez un module Node.js nommé `ec2_allocateaddress.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez un objet JSON contenant les paramètres utilisés pour attribuer une adresse IP Elastic qui, dans le cas présent, spécifie que le `Domain` est un VPC. Appelez la `allocateAddress` méthode de l'objet de service Amazon EC2.

Si l'appel aboutit, le paramètre `data` envoyé dans la fonction de rappel dispose d'une propriété `AllocationId` qui identifie l'adresse IP Elastic attribuée.

Créez un objet JSON pour les paramètres utilisés pour associer une adresse IP élastique à une instance Amazon EC2, y compris l'adresse `AllocationId` provenant de la nouvelle adresse attribuée et celle `InstanceId` de l'instance Amazon EC2. Appelez ensuite la `associateAddresses` méthode de l'objet de service Amazon EC2.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var paramsAllocateAddress = {
  Domain: "vpc",
};

// Allocate the Elastic IP address
ec2.allocateAddress(paramsAllocateAddress, function (err, data) {
  if (err) {
    console.log("Address Not Allocated", err);
  } else {
```

```
    console.log("Address allocated:", data.AllocationId);
    var paramsAssociateAddress = {
      AllocationId: data.AllocationId,
      InstanceId: "INSTANCE_ID",
    };
    // Associate the new Elastic IP address with an EC2 instance
    ec2.associateAddress(paramsAssociateAddress, function (err, data) {
      if (err) {
        console.log("Address Not Associated", err);
      } else {
        console.log("Address associated:", data.AssociationId);
      }
    });
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_allocateaddress.js
```

Cet exemple de code se trouve ici sur GitHub.

Libération d'une adresse IP Elastic

Créez un module Node.js nommé `ec2_releaseaddress.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon EC2, créez un objet de `AWS.EC2` service. Créez un objet JSON contenant les paramètres utilisés pour libérer une adresse IP Elastic qui, dans le cas présent, spécifie l'`AllocationId` de l'adresse IP Elastic. La publication d'une adresse IP élastique la dissocie également de toute instance Amazon EC2. Appelez la `releaseAddress` méthode de l'objet de service Amazon EC2.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var paramsReleaseAddress = {
  AllocationId: "ALLOCATION_ID",
};
```

```
// Disassociate the Elastic IP address from EC2 instance
ec2.releaseAddress(paramsReleaseAddress, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Address released");
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_releaseaddress.js
```

Cet exemple de code se trouve ici sur GitHub.

AWS Elemental MediaConvertExemples

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 Guide de l'utilisateur AWS Elemental MediaConvert.

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

Rubriques

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

Obtenir votre point de terminaison spécifique à une région pour MediaConvert

Cet exemple de code Node.js présente :

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

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 :

- `describeEndpoints`

> **Important**
>
> L'agent HTTP/HTTPS Node.js par défaut crée une nouvelle connexion TCP pour chaque nouvelle demande. Pour éviter les coûts liés à l'établissement d'une nouvelle connexion, les connexions AWS SDK for JavaScript TCP sont réutilisées. Pour plus d'informations, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- 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.

Obtention de l'URL du point de terminaison

Créez un module Node.js nommé `emc_getendpoint.js`. Veillez à configurer le kit SDK comme indiqué précédemment.

Créez un objet pour transmettre les paramètres de requête vides pour la `describeEndpoints` méthode de la classe `AWS.MediaConvert` client. Pour appeler la méthode `describeEndpoints`, créez une promesse pour appeler un objet de service MediaConvert , en transmettant les paramètres. Traitez la réponse dans le rappel de promesse.

```
// Load the SDK for JavaScript.
const aws = require("aws-sdk");

// Set the AWS Region.
aws.config.update({ region: "us-west-2" });

// Create the client.
const mediaConvert = new aws.MediaConvert({ apiVersion: "2017-08-29" });

exports.handler = async (event, context) => {
  // Create empty request parameters
  const params = {
    MaxResults: 0,
  };

  try {
    const { Endpoints } = await mediaConvert
      .describeEndpoints(params)
      .promise();
    console.log("Your MediaConvert endpoint is ", Endpoints);
  } catch (err) {
    console.log("MediaConvert Error", err);
  }
};
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node emc_getendpoint.js
```

Cet exemple de code se trouve ici sur 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.

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 :

- `createJob`
- `cancelJob`
- `getJob`
- `listJobs`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

- 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.

Configuration du kit SDK

Configurez le SDK pour JavaScript en créant un objet de configuration globale, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`. Étant donné qu'il MediaConvert utilise des points de terminaison personnalisés pour chaque compte, vous devez également configurer la classe `AWS.MediaConvert` client pour utiliser le point de terminaison spécifique à votre région. Pour ce faire, vous définissez le paramètre `endpoint` sur `AWS.config.mediaconvert`.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };
```

Définition d'une tâche de transcodage simple

Créez un module Node.js nommé `emc_createjob.js`. Veillez à configurer le kit SDK comme indiqué précédemment. 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.

```
var params = {
```

```
  Queue: "JOB_QUEUE_ARN",
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN",
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "s3://OUTPUT_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: "s3://INPUT_BUCKET_AND_FILE_NAME",
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};
```

Création d'une tâche de transcodage

Après avoir créé l'objet JSON des paramètres de tâche, appelez la méthode `createJob` en créant une promesse pour appeler un objet de service `AWS.MediaConvert` et en transmettant les paramètres. Traitez ensuite l'élément response dans le rappel de promesse. L'ID de la tâche créée est renvoyé dans les données `data` de la réponse.

```
// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .createJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Job created! ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node emc_createjob.js
```

Cet exemple de code se trouve ici sur GitHub.

Annulation d'une tâche de transcodage

Créez un module Node.js nommé `emc_canceljob.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Créez l'objet JSON qui inclut l'ID de la tâche à annuler. Appelez ensuite la méthode `cancelJob` en créant une promesse pour appeler un objet de service `AWS.MediaConvert`, en transmettant les paramètres. Traitez la réponse dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
```

```
// Set MediaConvert to customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Id: "JOB_ID" /* required */,
};

// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .cancelJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Job  " + params.Id + " is canceled");
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ec2_canceljob.js
```

Cet exemple de code se trouve ici sur GitHub.

Affichage de la liste des tâches de transcodage récentes

Créez un module Node.js nommé `emc_listjobs.js`. Veillez à configurer le kit SDK comme indiqué précédemment.

Créez l'objet JSON des paramètres, en incluant les valeurs qui spécifient si vous souhaitez trier la liste dans l'ordre `ASCENDING` (croissant) ou `DESCENDING` (décroissant), l'ARN de la file d'attente de tâches à vérifier et le statut des tâches à inclure. Appelez ensuite la méthode `listJobs` en créant une promesse pour appeler un objet de service `AWS.MediaConvert`, en transmettant les paramètres. Traitez la réponse dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
```

```
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED",
};

// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .listJobs(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Jobs: ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node emc_listjobs.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation de modèles de Job dans MediaConvert

Cet exemple de code Node.js présente :

- Comment créer des modèles de MediaConvert tâches.

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

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 :

- `createJobTemplate`
- `createJob`
- `deleteJobTemplate`
- `listJobTemplates`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- 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.

Création d'un modèle de tâche

Créez un module Node.js nommé `emc_create_jobtemplate.js`. Veillez à configurer le kit SDK comme indiqué précédemment.

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 méthode `createJobTemplate` en créant une promesse pour appeler un objet de service `AWS.MediaConvert`, en transmettant les paramètres. Traitez ensuite l'élément response dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Category: "YouTube Jobs",
  Description: "Final production transcode",
  Name: "DemoTemplate",
  Queue: "JOB_QUEUE_ARN",
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "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",
    },
  },
};

// Create a promise on a MediaConvert object
var templatePromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .createJobTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected status
templatePromise.then(
  function (data) {
    console.log("Success!", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node emc_create_jobtemplate.js
```

Cet exemple de code se trouve ici sur GitHub.

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

Créez un module Node.js nommé `emc_template_createjob.js`. Veillez à configurer le kit SDK comme indiqué précédemment.

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 méthode

`createJobs` en créant une promesse pour appeler un objet de service `AWS.MediaConvert`, en transmettant les paramètres. Traitez la réponse dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Queue: "QUEUE_ARN",
  JobTemplate: "TEMPLATE_NAME",
  Role: "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: "s3://BUCKET_NAME/FILE_NAME",
      },
    ],
  },
};

// Create a promise on a MediaConvert object
var templateJobPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
```

```
  .createJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
templateJobPromise.then(
  function (data) {
    console.log("Success! ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node emc_template_createjob.js
```

Cet exemple de code se trouve ici sur GitHub.

Liste de vos modèles de tâche

Créez un module Node.js nommé `emc_listtemplates.js`. Veillez à configurer le kit SDK comme indiqué précédemment.

Créez un objet afin de transmettre les paramètres de demande pour la méthode `listTemplates` de la classe client `AWS.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 `listTemplates` méthode, créez une promesse pour appeler un objet de MediaConvert service en transmettant les paramètres. Traitez ensuite l'élément response dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  ListBy: "NAME",
  MaxResults: 10,
```

```
  Order: "ASCENDING",
};

// Create a promise on a MediaConvert object
var listTemplatesPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .listJobTemplates(params)
  .promise();

// Handle promise's fulfilled/rejected status
listTemplatesPromise.then(
  function (data) {
    console.log("Success ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node emc_listtemplates.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un modèle de tâche

Créez un module Node.js nommé `emc_deletetemplate.js`. Veillez à configurer le kit SDK comme indiqué précédemment.

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

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };
```

```
var params = {
  Name: "TEMPLATE_NAME",
};

// Create a promise on a MediaConvert object
var deleteTemplatePromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .deleteJobTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected status
deleteTemplatePromise.then(
  function (data) {
    console.log("Success ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node emc_deletetemplate.js
```

Cet exemple de code se trouve ici sur GitHub.

Exemples d'Amazon S3 Glacier

Amazon S3 Glacier est un service de stockage cloud sécurisé pour l'archivage des données et la sauvegarde à long terme. Le service est optimisé pour les données auxquelles vous accédez de manière occasionnelle et pour lesquelles un délai d'extraction de plusieurs heures reste acceptable.

L' JavaScript API d'Amazon S3 Glacier est exposée par le biais de la classe `AWS.Glacier` client. Pour plus d'informations sur l'utilisation de la classe client S3 Glacier, consultez `Class: AWS.Glacier`la référence de l'API.

Rubriques

- Création d'un S3 Glacier Vault
- Téléchargement d'une archive sur S3 Glacier
- Effectuer un téléchargement partitionné vers S3 Glacier

Création d'un S3 Glacier Vault

Cet exemple de code Node.js présente :

- Comment créer un coffre-fort à l'aide de la `createVault` méthode de l'objet de service Amazon S3 Glacier.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Création du coffre

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

Téléchargement d'une archive sur S3 Glacier

Cet exemple de code Node.js présente :

- Comment télécharger une archive sur Amazon S3 Glacier à l'aide de la `uploadArchive` méthode de l'objet de service S3 Glacier.

L'exemple suivant télécharge un seul `Buffer` objet sous forme d'archive complète à l'aide de la `uploadArchive` méthode de l'objet de service S3 Glacier.

Pour cet exemple, on suppose que vous avez déjà créé un coffre nommé `YOUR_VAULT_NAME`. Le kit SDK calcule automatiquement le total de contrôle de hachage d'arborescence pour les données chargées, mais vous pouvez le remplacer en transmettant votre propre paramètre du total de contrôle :

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Chargement de l'archive

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

Effectuer un téléchargement partitionné vers S3 Glacier

L'exemple suivant crée un téléchargement partitionné à partir de segments de 1 mégaoctet d'un `Buffer` objet à l'aide de la méthode de `initiateMultipartUpload` l'objet de service Amazon S3 Glacier.

Pour cet exemple, on suppose que vous avez déjà créé un coffre nommé `YOUR_VAULT_NAME`. Un hachage d'arborescence SHA-256 complet est calculé manuellement à l'aide de la méthode `computeChecksums`.

```
// Create a new service object and some supporting variables
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" }),
  vaultName = "YOUR_VAULT_NAME",
  buffer = new Buffer(2.5 * 1024 * 1024), // 2.5MB buffer
  partSize = 1024 * 1024, // 1MB chunks,
  numPartsLeft = Math.ceil(buffer.length / partSize),
  startTime = new Date(),
  params = { vaultName: vaultName, partSize: partSize.toString() };

// Compute the complete SHA-256 tree hash so we can pass it
// to completeMultipartUpload request at the end
```

```
var treeHash = glacier.computeChecksums(buffer).treeHash;

// Initiate the multipart upload
console.log("Initiating upload to", vaultName);
// Call Glacier to initiate the upload.
glacier.initiateMultipartUpload(params, function (mpErr, multipart) {
  if (mpErr) {
    console.log("Error!", mpErr.stack);
    return;
  }
  console.log("Got upload ID", multipart.uploadId);

  // Grab each partSize chunk and upload it as a part
  for (var i = 0; i < buffer.length; i += partSize) {
    var end = Math.min(i + partSize, buffer.length),
      partParams = {
        vaultName: vaultName,
        uploadId: multipart.uploadId,
        range: "bytes " + i + "-" + (end - 1) + "/*",
        body: buffer.slice(i, end),
      };

    // Send a single part
    console.log("Uploading part", i, "=", partParams.range);
    glacier.uploadMultipartPart(partParams, function (multiErr, mData) {
      if (multiErr) return;
      console.log("Completed part", this.request.params.range);
      if (--numPartsLeft > 0) return; // complete only when all parts uploaded

      var doneParams = {
        vaultName: vaultName,
        uploadId: multipart.uploadId,
        archiveSize: buffer.length.toString(),
        checksum: treeHash, // the computed tree hash
      };

      console.log("Completing upload...");
      glacier.completeMultipartUpload(doneParams, function (err, data) {
        if (err) {
          console.log("An error occurred while uploading the archive");
          console.log(err);
        } else {
          var delta = (new Date() - startTime) / 1000;
          console.log("Completed upload in", delta, "seconds");
```

```
          console.log("Archive ID:", data.archiveId);
          console.log("Checksum:  ", data.checksum);
        }
      });
    });
  }
});
```

AWSExemples IAM

AWS Identity and Access Management(IAM) est un service Web qui permet aux clients d'Amazon Web Services de gérer les utilisateurs et les autorisations des utilisateurs dansAWS. Le service est destiné aux organisations ayant plusieurs utilisateurs ou des systèmes dans le cloud qui utilisent AWS des produits. Avec IAM, vous pouvez gérer de manière centralisée les utilisateurs, les informations d'identification de sécurité telles que les clés d'accès et les autorisations qui contrôlent les AWS ressources auxquelles les utilisateurs peuvent accéder.

L' JavaScript API pour IAM est exposée par le biais de la classe `AWS.IAM` client. Pour plus d'informations sur l'utilisation de la classe client IAM, consultez la référence `Class: AWS.IAM`de l'API.

Rubriques

- Gestion des utilisateurs IAM
- Utilisation des stratégies IAM
- Gestion des clés d’accès IAM
- Utilisation des certificats de serveur IAM
- Gestion des alias de compte IAM

Gestion des utilisateurs IAM

Cet exemple de code Node.js présente :

- Comment récupérer une liste d'utilisateurs IAM
- Procédure de création et de suppression des utilisateurs.
- Procédure de mise à jour d'un nom d'utilisateur.

Scénario

Dans cet exemple, une série de modules Node.js sont utilisés pour créer et gérer des utilisateurs dans IAM. Les modules Node.js utilisent le SDK pour créer, supprimer et mettre JavaScript à jour les utilisateurs à l'aide des méthodes suivantes de la classe `AWS.IAM` client :

- `createUser`
- `listUsers`
- `updateUser`
- `getUser`
- `deleteUser`

Pour plus d'informations sur les utilisateurs IAM, consultez la section Utilisateurs IAM dans le Guide de l'utilisateur IAM.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Création d'un utilisateur

Créez un module Node.js nommé `iam_createuser.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires, ce qui inclut le nom d'utilisateur que vous souhaitez attribuer au nouvel utilisateur en tant que paramètre de ligne de commande.

Appelez la méthode `getUser` de l'objet de service `AWS.IAM` afin de vérifier si le nom d'utilisateur existe déjà. Si le nom d'utilisateur n'existe pas, appelez la méthode `createUser` pour le créer. Si le nom existe déjà, écrivez un message à cette fin à destination de la console.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_createuser.js USER_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Liste des utilisateurs de votre compte

Créez un module Node.js nommé `iam_listusers.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour répertorier vos utilisateurs et limitez le nombre de résultats renvoyés en définissant le paramètre `MaxItems` sur 10. Appelez la méthode `listUsers` de l'objet de service `AWS.IAM`. Écrivez le nom du premier utilisateur et sa date de création sur la console.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_listusers.js
```

Cet exemple de code se trouve ici sur GitHub.

Mise à jour du nom d'un utilisateur

Créez un module Node.js nommé `iam_updateuser.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour répertorier les utilisateurs, en spécifiant le nom d'utilisateur actuel et le nouveau nom d'utilisateur en tant que paramètres de ligne de commande. Appelez la méthode `updateUser` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit sur la ligne de commande, en spécifiant le nom actuel de l'utilisateur, suivi de son nouveau nom.

```
node iam_updateuser.js ORIGINAL_USERNAME NEW_USERNAME
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un utilisateur

Créez un module Node.js nommé `iam_deleteuser.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet

JSON contenant les paramètres nécessaires, ce qui inclut le nom d'utilisateur que vous souhaitez supprimer en tant que paramètre de ligne de commande.

Appelez la méthode `getUser` de l'objet de service `AWS.IAM` afin de vérifier si le nom d'utilisateur existe déjà. Si le nom d'utilisateur n'existe pas déjà, écrivez un message à cette fin à destination à la console. Si l'utilisateur existe, appelez la méthode `deleteUser` pour le supprimer.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_deleteuser.js USER_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des stratégies IAM

Cet exemple de code Node.js présente :

- Comment créer et supprimer des politiques IAM
- Comment associer et détacher les politiques IAM des rôles.

Scénario

Vous accordez des autorisations à un utilisateur en créant une stratégie, à savoir un document qui répertorie les actions qu'un utilisateur peut réaliser et les ressources que ces actions peuvent concerner. Les actions ou ressources qui ne sont pas explicitement autorisées sont refusées par défaut. Vous pouvez créer des stratégies et les attacher à des utilisateurs, à des groupes d'utilisateurs, à des rôles pris en charge par des utilisateurs et à des ressources.

Dans cet exemple, une série de modules Node.js sont utilisés pour gérer les politiques dans IAM. Les modules Node.js utilisent le SDK pour JavaScript créer et supprimer des politiques, ainsi que pour associer et détacher des politiques de rôle à l'aide des méthodes suivantes de la classe `AWS.IAM` client :

- `createPolicy`
- `getPolicy`
- `listAttachedRolePolicies`
- `attachRolePolicy`
- `detachRolePolicy`

Pour plus d'informations sur les utilisateurs IAM, voir Présentation de la gestion des accès : autorisations et politiques dans le guide de l'utilisateur IAM.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créez un rôle IAM auquel vous pouvez associer des politiques. Pour plus d'informations sur la création de rôles, consultez la section Création de rôles IAM dans le guide de l'utilisateur IAM.

Création d'une politique IAM

Créez un module Node.js nommé `iam_createpolicy.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez deux objets JSON, l'un contenant le document de stratégie à créer et l'autre contenant les paramètres requis pour créer la stratégie, ce qui inclut l'objet JSON de la stratégie et le nom que vous voulez attribuer à la stratégie. Veillez à traiter le paramètre « stringify » de l'objet JSON de la stratégie dans les paramètres. Appelez la méthode `createPolicy` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_createpolicy.js
```

Cet exemple de code se trouve ici sur GitHub.

Obtenir une politique IAM

Créez un module Node.js nommé `iam_getpolicy.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour récupérer une stratégie, qui est l'ARN de la stratégie que vous souhaitez obtenir. Appelez la méthode `getPolicy` de l'objet de service `AWS.IAM`. Écrivez la description de stratégie sur la console.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_getpolicy.js
```

Cet exemple de code se trouve ici sur GitHub.

Attachement d'une stratégie de rôle gérée

Créez un module Node.js nommé `iam_attachrolepolicy.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour obtenir une liste des politiques IAM gérées associées à un rôle, qui comprend le nom du rôle. Indiquez le nom de rôle en tant que paramètre de ligne de commande. Appelez la méthode `listAttachedRolePolicies` de l'objet de service `AWS.IAM`, qui renvoie un tableau des stratégies gérées à la fonction de rappel.

Consultez la liste des membres du tableau afin de vérifier si la stratégie que vous souhaitez attacher au rôle est déjà attachée. Si elle ne l'est pas, appelez la méthode `attachRolePolicy` pour l'attacher.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_attachrolepolicy.js IAM_ROLE_NAME
```

Détachement d'une stratégie de rôle gérée

Créez un module Node.js nommé `iam_detachrolepolicy.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour obtenir une liste des politiques IAM gérées associées à un rôle, qui comprend le nom du rôle. Indiquez le nom de rôle en tant que paramètre de ligne de commande. Appelez la méthode `listAttachedRolePolicies` de l'objet de service `AWS.IAM`, qui renvoie un tableau des stratégies gérées dans la fonction de rappel.

Consultez la liste des membres du tableau afin de vérifier si la stratégie que vous souhaitez détacher du rôle est déjà attachée. Si elle l'est, appelez la méthode `detachRolePolicy` pour la détacher.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_detachrolepolicy.js IAM_ROLE_NAME
```

Gestion des clés d’accès IAM

Cet exemple de code Node.js présente :

- Procédure de gestion des clés d'accès de vos utilisateurs.

Scénario

Les utilisateurs ont besoin de leurs propres clés d'accès pour effectuer des appels programmatiques AWS depuis le SDK pour. JavaScript Pour ce faire, vous pouvez créer, modifier, afficher ou faire tourner les clés d'accès (ID de clé d'accès et clés d'accès secrètes) pour les utilisateurs IAM. Par défaut, lorsque vous créez une clé d'accès, son statut est `Active`, ce qui signifie que l'utilisateur peut l'utiliser pour les appels d'API.

Dans cet exemple, une série de modules Node.js sont utilisés pour gérer les clés d'accès dans IAM. Les modules Node.js utilisent le SDK pour gérer les clés JavaScript d'accès IAM à l'aide des méthodes suivantes de la classe `AWS.IAM` client :

- createAccessKey
- listAccessKeys
- getAccessKeyLastUsed
- updateAccessKey
- deleteAccessKey

Pour plus d'informations sur les clés d'accès IAM, consultez la section Clés d'accès dans le guide de l'utilisateur IAM.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.

- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Création de clés d'accès pour un utilisateur

Créez un module Node.js nommé `iam_createaccesskeys.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour créer de nouvelles clés d'accès, y compris le nom de l'utilisateur IAM. Appelez la méthode `createAccessKey` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande. Veillez à diriger les données renvoyées vers un fichier texte afin de ne pas perdre la clé secrète, qui peut être fournie une seule fois.

```
node iam_createaccesskeys.js > newuserkeys.txt
```

Cet exemple de code se trouve ici sur GitHub.

Liste des clés d'accès d'un utilisateur

Créez un module Node.js nommé `iam_listaccesskeys.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour récupérer les clés d'accès de l'utilisateur,

y compris le nom de l'utilisateur IAM et éventuellement le nombre maximum de paires de clés d'accès que vous souhaitez répertorier. Appelez la méthode `listAccessKeys` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_listaccesskeys.js
```

Cet exemple de code se trouve ici sur GitHub.

Obtention de la date à laquelle une clé d'accès a été utilisée pour la dernière fois

Créez un module Node.js nommé `iam_accesskeylastused.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres requis pour créer des clés d'accès, qui est l'ID de clé d'accès pour lequel vous souhaitez récupérer les informations sur la dernière utilisation. Appelez la méthode `getAccessKeyLastUsed` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_accesskeylastused.js
```

Cet exemple de code se trouve ici sur GitHub.

Mise à jour du statut des clés d'accès

Créez un module Node.js nommé `iam_updateaccesskey.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres requis pour mettre à jour le statut des clés d'accès, ce qui inclut l'ID de clé d'accès et le statut mis à jour. Le statut peut être `Active` ou `Inactive`. Appelez la méthode `updateAccessKey` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_updateaccesskey.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression des clés d'accès

Créez un module Node.js nommé `iam_deleteaccesskey.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres requis pour supprimer des clés d'accès, ce qui inclut l'ID des clés d'accès et le nom de l'utilisateur. Appelez la méthode `deleteAccessKey` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_deleteaccesskey.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des certificats de serveur IAM

Cet exemple de code Node.js présente :

- Procédure permettant d'effectuer des tâches de base de gestion des certificats de serveur pour les connexions HTTPS.

Scénario

Pour activer les connexions HTTPS à votre site Web ou à votre applicationAWS, vous avez besoin d'un certificat de serveur SSL/TLS. Pour utiliser un certificat que vous avez obtenu auprès d'un fournisseur externe avec votre site Web ou votre applicationAWS, vous devez télécharger le certificat dans IAM ou l'importer dans AWS Certificate Manager.

Dans cet exemple, une série de modules Node.js sont utilisés pour gérer les certificats de serveur dans IAM. Les modules Node.js utilisent le SDK pour gérer les certificats de serveur JavaScript à l'aide des méthodes suivantes de la classe `AWS.IAM` client :

- listServerCertificates
- getServerCertificate
- updateServerCertificate
- deleteServerCertificate

Pour plus d'informations sur les certificats de serveur, consultez la section Utilisation des certificats de serveur dans le guide de l'utilisateur IAM.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Liste des certificats de serveur

Créez un module Node.js nommé `iam_listservercerts.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Appelez la méthode `listServerCertificates` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_listservercerts.js
```

Cet exemple de code se trouve ici sur GitHub.

Obtention d'un certificat de serveur

Créez un module Node.js nommé `iam_getservercert.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour obtenir un certificat, ce qui inclut le nom du certificat de serveur que vous recherchez. Appelez la méthode `getServerCertificates` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_getservercert.js
```

Cet exemple de code se trouve ici sur GitHub.

Mise à jour d'un certificat de serveur

Créez un module Node.js nommé `iam_updateservercert.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres requis pour mettre à jour un certificat, ce qui comprend le nom du certificat de serveur existant, ainsi que le nom du nouveau certificat. Appelez la méthode `updateServerCertificate` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_updateservercert.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un certificat de serveur

Créez un module Node.js nommé `iam_deleteservercert.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres nécessaires pour supprimer un certificat de serveur, ce qui inclut le nom du certificat à supprimer. Appelez la méthode `deleteServerCertificates` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_deleteservercert.js
```

Cet exemple de code se trouve ici sur GitHub.

Gestion des alias de compte IAM

Cet exemple de code Node.js présente :

- Comment gérer les alias de votre identifiant de AWS compte.

Scénario

Si vous souhaitez que l'URL de votre page de connexion contienne le nom de votre entreprise ou un autre identifiant convivial au lieu de votre identifiant de AWS compte, vous pouvez créer un alias pour votre identifiant de AWS compte. Si vous créez un alias de compte AWS, l'URL de votre page de connexion est modifiée de manière à intégrer l'alias.

Dans cet exemple, une série de modules Node.js sont utilisés pour créer et gérer des alias de comptes IAM. Les modules Node.js utilisent le SDK pour gérer les alias JavaScript à l'aide des méthodes suivantes de la classe `AWS.IAM` client :

- createAccountAlias
- listAccountAliases
- deleteAccountAlias

Pour plus d'informations sur les alias de compte IAM, consultez la section Votre identifiant de AWS compte et son alias dans le guide de l'utilisateur IAM.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Création d'un alias de compte

Créez un module Node.js nommé `iam_createaccountalias.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres requis pour créer un alias de compte, ce qui inclut l'alias que vous voulez créer. Appelez la méthode `createAccountAlias` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_createaccountalias.js ALIAS
```

Cet exemple de code se trouve ici sur GitHub.

Liste des alias de compte

Créez un module Node.js nommé `iam_listaccountaliases.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres requis pour répertorier les alias de compte, ce qui inclut le nombre maximal d'éléments à renvoyer. Appelez la méthode `listAccountAliases` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_listaccountaliases.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un alias de compte

Créez un module Node.js nommé `iam_deleteaccountalias.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à IAM, créez un objet `AWS.IAM` de service. Créez un objet JSON contenant les paramètres requis pour supprimer un alias de compte, ce qui inclut

l'alias que vous voulez supprimer. Appelez la méthode `deleteAccountAlias` de l'objet de service `AWS.IAM`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node iam_deleteaccountalias.js ALIAS
```

Cet exemple de code se trouve ici sur GitHub.

Exemple Amazon Kinesis

Amazon Kinesis est une plateforme de diffusion de données qui propose de puissants services de chargement et d'analyse des données de streaming, ainsi que la possibilité de créer des applications de données de streaming personnalisées répondant à des besoins spécifiques. AWS

L' JavaScript API pour Kinesis est exposée par le biais de la classe `AWS.Kinesis` client. Pour plus d'informations sur l'utilisation de la classe client Kinesis, consultez la référence `Class: AWS.Kinesis`de l'API.

Rubriques

- Capture de la progression du défilement des pages Web avec Amazon Kinesis

Capture de la progression du défilement des pages Web avec Amazon Kinesis

Cet exemple de script de navigateur présente :

- Comment enregistrer la progression du défilement d'une page Web avec Amazon Kinesis comme exemple de statistiques d'utilisation d'une page de streaming pour une analyse ultérieure.

Scénario

Dans cet exemple, une simple page HTML simule le contenu d'une page de blog. Lorsque le lecteur fait défiler le billet de blog simulé, le script du navigateur utilise le SDK pour JavaScript enregistrer la distance de défilement vers le bas de la page et envoyer ces données à Kinesis en utilisant la méthode `putRecords`de la classe client Kinesis. Les données de streaming capturées par Amazon Kinesis Data Streams peuvent ensuite être traitées par les instances Amazon EC2 et stockées dans l'un des nombreux magasins de données, notamment Amazon DynamoDB et Amazon Redshift.

Tâches prérequises

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

- Créez un flux Kinesis. Vous devez inclure l'ARN de la ressource du flux dans le script du navigateur. Pour plus d'informations sur la création d'Amazon Kinesis Data Streams, consultez Managing Kinesis Streams dans le manuel Amazon Kinesis Data Streams Developer Guide.
- Créez un pool d'identités Amazon Cognito dont l'accès est activé pour les identités non authentifiées. Vous devez inclure l'ID du groupe d'identités dans le code afin d'obtenir les

informations d'identification relatives au script du navigateur. Pour plus d'informations sur les groupes d'identités Amazon Cognito, consultez la section Groupes d'identités du manuel Amazon Cognito Developer Guide.

- Créez un rôle IAM dont la politique autorise l'envoi de données vers un flux Kinesis. Pour plus d'informations sur la création d'un rôle IAM, consultez la section Création d'un rôle pour déléguer des autorisations à un AWS service dans le Guide de l'utilisateur IAM.

Utilisez la stratégie de rôle suivante lors de la création du rôle IAM.

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

Page du blog

Le code HTML de la page du blog se compose principalement d'une série de paragraphes contenus dans un élément `<div>`. La hauteur de défilement de cet élément `<div>` sert à calculer jusqu'où un lecteur a déroulé le contenu qu'il lit. Le code HTML contient également une paire d'éléments

`<script>`. L'un de ces éléments ajoute le SDK JavaScript pour la page et l'autre ajoute le script de navigateur qui enregistre la progression du défilement sur la page et la rapporte à Kinesis.

```
<!DOCTYPE html>
<html>
    <head>
        <title>AWS SDK for JavaScript - Amazon Kinesis Application</title>
    </head>
    <body>
        <div id="BlogContent" style="width: 60%; height: 800px; overflow: auto;margin:
 auto; text-align: center;">
            <div>
                <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum
 vitae nulla eget nisl bibendum feugiat. Fusce rhoncus felis at ultricies luctus.
 Vivamus fermentum cursus sem at interdum. Proin vel lobortis nulla. Aenean rutrum
 odio in tellus semper rhoncus. Nam eu felis ac augue dapibus laoreet vel in erat.
 Vivamus vitae mollis turpis. Integer sagittis dictum odio. Duis nec sapien diam.
 In imperdiet sem nec ante laoreet, vehicula facilisis sem placerat. Duis ut metus
 egestas, ullamcorper neque et, accumsan quam. Class aptent taciti sociosqu ad litora
 torquent per conubia nostra, per inceptos himenaeos.
                </p>
                <!-- Additional paragraphs in the blog page appear here -->
            </div>
        </div>
        <script src="https://sdk.amazonaws.com/js/aws-sdk-2.283.1.min.js"></script>
        <script src="kinesis-example.js"></script>
    </body>
</html>
```

Configuration du kit SDK

Obtenez les informations d'identification nécessaires pour configurer le SDK en appelant la `CognitoIdentityCredentials` méthode, en fournissant l'ID du pool d'identités Amazon Cognito. En cas de succès, créez l'objet de service Kinesis dans la fonction de rappel.

L'extrait de code suivant illustre cette étape. (Pour obtenir l'exemple complet, consultez Code de capture de la progression du défilement d'une page web.)

```
// Configure Credentials to use Cognito
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
```

```
});

AWS.config.region = "REGION";
// We're going to partition Amazon Kinesis records based on an identity.
// We need to get credentials first, then attach our event listeners.
AWS.config.credentials.get(function (err) {
  // attach event listener
  if (err) {
    alert("Error retrieving credentials.");
    console.error(err);
    return;
  }
  // create Amazon Kinesis service object
  var kinesis = new AWS.Kinesis({
    apiVersion: "2013-12-02",
  });
```

Création des enregistrements de défilement

La progression du défilement est calculée à l'aide des propriétés `scrollHeight` et `scrollTop` de l'élément `<div>` contenant le contenu du billet de blog. Chaque enregistrement de défilement est créé dans une fonction d'écoute d'événements pour l'`scroll`événement, puis ajouté à un tableau d'enregistrements pour être soumis périodiquement à Kinesis.

L'extrait de code suivant illustre cette étape. (Pour obtenir l'exemple complet, consultez Code de capture de la progression du défilement d'une page web.)

```
  // Get the ID of the Web page element.
  var blogContent = document.getElementById("BlogContent");

  // Get Scrollable height
  var scrollableHeight = blogContent.clientHeight;

  var recordData = [];
  var TID = null;
  blogContent.addEventListener("scroll", function (event) {
    clearTimeout(TID);
    // Prevent creating a record while a user is actively scrolling
    TID = setTimeout(function () {
      // calculate percentage
      var scrollableElement = event.target;
      var scrollHeight = scrollableElement.scrollHeight;
      var scrollTop = scrollableElement.scrollTop;
```

```
    var scrollTopPercentage = Math.round((scrollTop / scrollHeight) * 100);
    var scrollBottomPercentage = Math.round(
      ((scrollTop + scrollableHeight) / scrollHeight) * 100
    );

    // Create the Amazon Kinesis record
    var record = {
      Data: JSON.stringify({
        blog: window.location.href,
        scrollTopPercentage: scrollTopPercentage,
        scrollBottomPercentage: scrollBottomPercentage,
        time: new Date(),
      }),
      PartitionKey: "partition-" + AWS.config.credentials.identityId,
    };
    recordData.push(record);
  }, 100);
});
```

Soumission d'enregistrements à Kinesis

Une fois par seconde, s'il y a des enregistrements dans le tableau, ces enregistrements en attente sont envoyés à Kinesis.

L'extrait de code suivant illustre cette étape. (Pour obtenir l'exemple complet, consultez Code de capture de la progression du défilement d'une page web.)

```
// upload data to Amazon Kinesis every second if data exists
setInterval(function () {
  if (!recordData.length) {
    return;
  }
  // upload data to Amazon Kinesis
  kinesis.putRecords(
    {
      Records: recordData,
      StreamName: "NAME_OF_STREAM",
    },
    function (err, data) {
      if (err) {
        console.error(err);
      }
```

```
      }
    );
    // clear record data
    recordData = [];
  }, 1000);
});
```

Code de capture de la progression du défilement d'une page web

Voici le code du script de navigateur pour l'exemple de progression du défilement d'une page Web par capture Kinesis.

```
// Configure Credentials to use Cognito
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

AWS.config.region = "REGION";
// We're going to partition Amazon Kinesis records based on an identity.
// We need to get credentials first, then attach our event listeners.
AWS.config.credentials.get(function (err) {
  // attach event listener
  if (err) {
    alert("Error retrieving credentials.");
    console.error(err);
    return;
  }
  // create Amazon Kinesis service object
  var kinesis = new AWS.Kinesis({
    apiVersion: "2013-12-02",
  });

  // Get the ID of the Web page element.
  var blogContent = document.getElementById("BlogContent");

  // Get Scrollable height
  var scrollableHeight = blogContent.clientHeight;

  var recordData = [];
  var TID = null;
  blogContent.addEventListener("scroll", function (event) {
    clearTimeout(TID);
    // Prevent creating a record while a user is actively scrolling
```

```
    TID = setTimeout(function () {
      // calculate percentage
      var scrollableElement = event.target;
      var scrollHeight = scrollableElement.scrollHeight;
      var scrollTop = scrollableElement.scrollTop;

      var scrollTopPercentage = Math.round((scrollTop / scrollHeight) * 100);
      var scrollBottomPercentage = Math.round(
        ((scrollTop + scrollableHeight) / scrollHeight) * 100
      );

      // Create the Amazon Kinesis record
      var record = {
        Data: JSON.stringify({
          blog: window.location.href,
          scrollTopPercentage: scrollTopPercentage,
          scrollBottomPercentage: scrollBottomPercentage,
          time: new Date(),
        }),
        PartitionKey: "partition-" + AWS.config.credentials.identityId,
      };
      recordData.push(record);
    }, 100);
  });

  // upload data to Amazon Kinesis every second if data exists
  setInterval(function () {
    if (!recordData.length) {
      return;
    }
    // upload data to Amazon Kinesis
    kinesis.putRecords(
      {
        Records: recordData,
        StreamName: "NAME_OF_STREAM",
      },
      function (err, data) {
        if (err) {
          console.error(err);
        }
      }
    );
    // clear record data
    recordData = [];
```

```
  }, 1000);
});
```

Exemples Amazon S3

Amazon Simple Storage Service (Amazon S3) est un service Web qui fournit un stockage dans le cloud hautement évolutif. Amazon S3 fournit un stockage d'objets facile à utiliser, avec une interface de service Web simple permettant de stocker et de récupérer n'importe quel volume de données, où que vous soyez sur le Web.

L' JavaScript API d'Amazon S3 est exposée par le biais de la classe `AWS.S3` client. Pour plus d'informations sur l'utilisation de la classe client Amazon S3, consultez `Class: AWS.S3`la référence de l'API.

Rubriques

- Exemples de navigateurs Amazon S3
- Exemples Node.js d'Amazon S3

Exemples de navigateurs Amazon S3

Les rubriques suivantes présentent deux exemples de la manière dont le navigateur AWS SDK for JavaScript peut être utilisé pour interagir avec les compartiments Amazon S3.

- La première montre un scénario simple dans lequel les photos existantes dans un compartiment Amazon S3 peuvent être consultées par n'importe quel utilisateur (non authentifié).
- La seconde montre un scénario plus complexe dans lequel les utilisateurs sont autorisés à effectuer des opérations sur les photos du bucket, telles que le téléchargement, la suppression, etc.

Rubriques

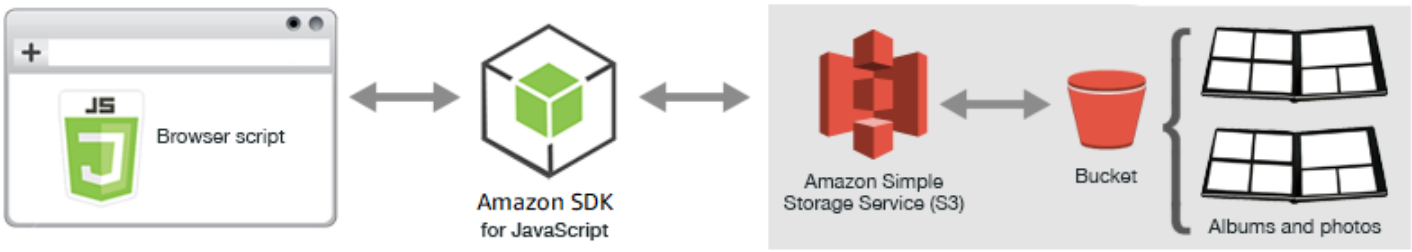
Affichage de photos dans un compartiment Amazon S3 à partir d'un navigateur

Cet exemple de code de script de navigateur illustre :

- Comment créer un album photo dans un compartiment Amazon Simple Storage Service (Amazon S3) et autoriser les utilisateurs non authentifiés à voir les photos.

Scénario

Dans cet exemple, une simple page HTML fournit une application basée sur un navigateur pour afficher des photos dans un album photo. L'album photo se trouve dans un compartiment Amazon S3 dans lequel les photos sont téléchargées.



Le script de navigateur utilise le SDK JavaScript pour interagir avec un compartiment Amazon S3. Le script utilise la `listObjects`méthode de la classe client Amazon S3 pour vous permettre de visualiser les albums photos.

Tâches prérequises

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

> **Note**
>
> Dans cet exemple, vous devez utiliser la même AWS région pour le compartiment Amazon S3 et le pool d'identités Amazon Cognito.

Créer le compartiment

Dans la console Amazon S3, créez un compartiment Amazon S3 dans lequel vous pouvez stocker des albums et des photos. Pour plus d'informations sur l'utilisation de la console pour créer un compartiment S3, consultez la section Création d'un compartiment dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Lorsque vous créez le compartiment S3, veillez à :

- Noter le nom du compartiment afin de pouvoir l'utiliser ultérieurement dans la tâche prérequise Configurer des autorisations de rôle.
- Choisissez une AWS région dans laquelle créer le compartiment. Il doit s'agir de la même région que celle que vous utiliserez pour créer un pool d'identités Amazon Cognito lors d'une tâche préalable ultérieure, Créer un pool d'identités.
- Configurez les autorisations du bucket en suivant la procédure de définition des autorisations pour l'accès au site Web dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Créer un groupe d'identités

Dans la console Amazon Cognito, créez un pool d'identités Amazon Cognito, comme décrit the section called “Étape 1 : créer un pool d'identités Amazon Cognito” dans la rubrique Getting Started in a Browser Script.

Lorsque vous créez le pool d'identités, notez le nom du pool d'identités, ainsi que le nom du rôle de l'identité non authentifiée.

Configurer les autorisations de rôle

Pour autoriser l'affichage d'albums et de photos, vous devez ajouter des autorisations à un rôle IAM du pool d'identités que vous venez de créer. Commencez par créer une stratégie comme suit.

1. Ouvrez la console IAM.

2. Dans le volet de navigation de gauche, choisissez Policies (Stratégies), puis le bouton Create policy (Créer une stratégie).
3. Dans l'onglet JSON, entrez la définition JSON suivante en remplaçant BUCKET_NAME par le nom de votre compartiment.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME"
         ]
      }
   ]
}
```

4. Choisissez le bouton Review policy (Vérifier la stratégie), nommez la stratégie et fournissez une description (si vous le souhaitez), puis choisissez le bouton Create policy (Créer la stratégie).

   N'oubliez pas de noter le nom afin de pouvoir le retrouver et de l'associer ultérieurement au rôle IAM.

Une fois la politique créée, revenez à la console IAM. Recherchez le rôle IAM pour l'identité non authentifiée créée par Amazon Cognito dans la tâche préalable précédente, Créer un pool d'identités. Utilisez la stratégie que vous venez de créer pour ajouter des autorisations à cette identité.

Bien que le flux de travail pour cette tâche soit généralement le même que celui de the section called “Étape 2 : ajouter une politique au rôle IAM créé” de la rubrique Démarrage dans un script de navigateur, quelques différences doivent être notées :

- Utilisez la nouvelle politique que vous venez de créer, et non une politique pour Amazon Polly.
- Sur la page Attach Permissions (Attacher des autorisations), pour retrouver rapidement la nouvelle stratégie, ouvrez la liste Filter policies (Filtrer les stratégies) et choisissez Customer managed (Gérées par l’utilisateur).

Pour plus d'informations sur la création d'un rôle IAM, consultez la section Création d'un rôle pour déléguer des autorisations à un AWS service dans le Guide de l'utilisateur IAM.

Configurer CORS

Avant que le script du navigateur puisse accéder au compartiment Amazon S3, vous devez configurer sa configuration CORS comme suit.

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

JSON

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

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>HEAD</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

Créer des albums et charge des photos

Cet exemple permettant uniquement aux utilisateurs d'afficher les photos qui sont déjà dans le compartiment, vous devez créer des albums dans le compartiment et y charger des photos.

> **Note**
>
> Dans cet exemple, les noms de fichier photo doivent commencer par un trait de soulignement (« _ ») unique. Ce caractère sera important ultérieurement pour le filtrage. Veillez également à respecter les droits d'auteur des propriétaires des photos.

1. Dans la console Amazon S3, ouvrez le compartiment que vous avez créé précédemment.
2. Dans l'onglet Overview (Présentation), cliquez sur le bouton Create folder (Créer un dossier) pour créer des dossiers. Pour cet exemple, nommez les dossiers « album1 », « album2 » et « album3 ».
3. Pour album1 puis album2, sélectionnez le dossier et chargez des photos comme suit :
   a. Choisissez le bouton Upload (Charger).
   b. Faites glisser ou choisissez les fichiers photos que vous souhaitez utiliser, puis choisissez Next (Suivant).
   c. Sous Manage public permissions Gérer les autorisations publiques), choisissez Grant public read access to this object(s) (Octroyer un accès en lecture public à ces objets).
   d. Choisissez le bouton Upload (Charger)(dans le coin inférieur gauche).
4. Laissez album3 vide.

Définition de la page Web

Le code HTML pour l'application d'affichage de photos se compose d'un élément `<div>` dans lequel le script de navigateur crée l'interface d'affichage. Le premier élément `<script>` ajoute le kit SDK au script de navigateur. Le deuxième `<script>` élément ajoute le JavaScript fichier externe qui contient le code du script du navigateur.

Pour cet exemple, le fichier est nommé `PhotoViewer.js`, et se trouve dans le même dossier que le fichier HTML. Pour trouver le SDK_VERSION_NUMBER actuel, consultez la référence d'API du SDK pour le guide de référence des API. JavaScript AWS SDK for JavaScript

```
<!DOCTYPE html>
<html>
```

```
  <head>
    <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./PhotoViewer.js"></script>
    <script>listAlbums();</script>
  </head>
  <body>
    <h1>Photo Album Viewer</h1>
    <div id="viewer" />
  </body>
</html>
```

Configuration du kit SDK

Obtenez les informations d'identification dont vous avez besoin pour configurer le kit SDK en appelant la méthode `CognitoIdentityCredentials`. Vous devez fournir l'ID du pool d'identités Amazon Cognito. Ensuite, créez un objet de service `AWS.S3`.

```
// **DO THIS**:
//   Replace BUCKET_NAME with the bucket name.
//
var albumBucketName = "BUCKET_NAME";

// **DO THIS**:
//   Replace this block of code with the sample code located at:
//   Cognito -- Manage Identity Pools -- [identity_pool_name] -- Sample Code --
 JavaScript
//
// Initialize the Amazon Cognito credentials provider
AWS.config.region = "REGION"; // Region
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

// Create a new service object
var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

// A utility function to create HTML.
```

```
function getHtml(template) {
  return template.join("\n");
}
```

Dans cet exemple, le reste du code définit les fonctions suivantes pour collecter et présenter des informations sur les albums et les photos du compartiment.

- `listAlbums`
- `viewAlbum`

Liste des albums dans le compartiment

Pour répertorier tous les albums existants dans le compartiment, la fonction `listAlbums` de l'application appelle la méthode `listObjects` de l'objet de service `AWS.S3`. La fonction utilise la propriété `CommonPrefixes` de sorte que l'appel renvoie uniquement les objets utilisés en tant qu'albums (c'est-à-dire, les dossiers).

Le reste de la fonction prend la liste des albums du compartiment Amazon S3 et génère le code HTML nécessaire pour afficher la liste des albums sur la page Web.

```
// List the photo albums that exist in the bucket.
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          '<button style="margin:5px;" onclick="viewAlbum(\'' +
            albumName +
            "')\">",
          albumName,
          "</button>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml(["<p>Click on an album name to view it.</p>"])
```

```
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
      ];
      document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    }
  });
}
```

Affichage d'un album

Pour afficher le contenu d'un album dans le compartiment Amazon S3, la `viewAlbum` fonction de l'application prend un nom d'album et crée la clé Amazon S3 pour cet album. Ensuite, la fonction appelle la méthode `listObjects` de l'objet de service `AWS.S3` pour obtenir une liste de tous les objets (les photos) de l'album.

Le reste de la fonction prend la liste des objets (photos) de l'album et génère le code HTML nécessaire à l'affichage des photos dans la page web.

```
// Show the photos that exist in an album.
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Request instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        "<br/>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
```

```
        "<div>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>The following photos are present.</p>"
      : "<p>There are no photos in this album.</p>";
    var htmlTemplate = [
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      "<h2>",
      "End of Album: " + albumName,
      "</h2>",
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
    ];
    document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    document
      .getElementsByTagName("img")[0]
      .setAttribute("style", "display:none;");
  });
}
```

Affichage de photos dans un compartiment Amazon S3 : code complet

Cette section contient le code HTML complet et le JavaScript code de l'exemple dans lequel les photos d'un compartiment Amazon S3 peuvent être visualisées. Consultez la section parente pour plus de détails et les conditions préalables.

Code HTML pour l'exemple :

```
<!DOCTYPE html>
<html>
  <head>
    <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./PhotoViewer.js"></script>
    <script>listAlbums();</script>
  </head>
  <body>
    <h1>Photo Album Viewer</h1>
    <div id="viewer" />
  </body>
</html>
```

Cet exemple de code se trouve ici sur GitHub.

Code de script de navigateur pour l'exemple :

```
//
// Data constructs and initialization.
//

// **DO THIS**:
//   Replace BUCKET_NAME with the bucket name.
//
var albumBucketName = "BUCKET_NAME";

// **DO THIS**:
//   Replace this block of code with the sample code located at:
//   Cognito -- Manage Identity Pools -- [identity_pool_name] -- Sample Code --
 JavaScript
//
// Initialize the Amazon Cognito credentials provider
AWS.config.region = "REGION"; // Region
```

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

// Create a new service object
var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

// A utility function to create HTML.
function getHtml(template) {
  return template.join("\n");
}

//
// Functions
//

// List the photo albums that exist in the bucket.
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          '<button style="margin:5px;" onclick="viewAlbum(\'' +
            albumName +
            "')\">",
          albumName,
          "</button>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml(["<p>Click on an album name to view it.</p>"])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
```

```
        "<ul>",
        getHtml(albums),
        "</ul>",
      ];
      document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    }
  });
}

// Show the photos that exist in an album.
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Request instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        "<br/>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>The following photos are present.</p>"
      : "<p>There are no photos in this album.</p>";
    var htmlTemplate = [
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
```

```
      "</button>",
      "</div>",
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      "<h2>",
      "End of Album: " + albumName,
      "</h2>",
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
    ];
    document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    document
      .getElementsByTagName("img")[0]
      .setAttribute("style", "display:none;");
  });
}
```

Cet exemple de code se trouve ici sur GitHub.

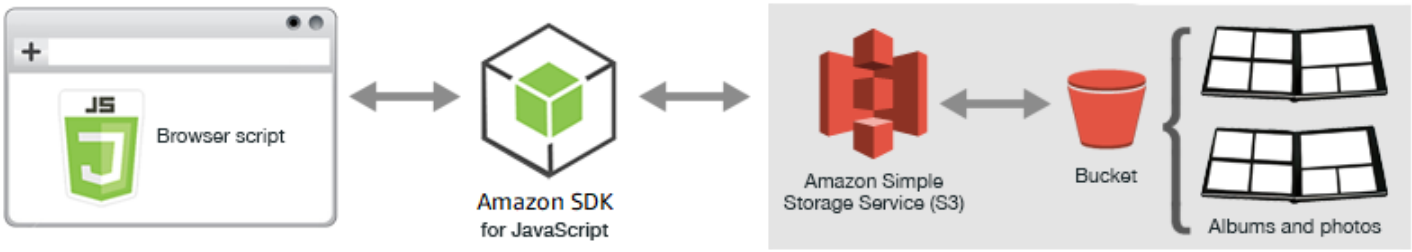
Téléchargement de photos vers Amazon S3 depuis un navigateur

Cet exemple de code de script de navigateur illustre :

- Comment créer une application de navigateur qui permet aux utilisateurs de créer des albums photos dans un compartiment Amazon S3 et de télécharger des photos dans les albums.

Scénario

Dans cet exemple, une simple page HTML fournit une application basée sur un navigateur pour créer des albums photos dans un compartiment Amazon S3 dans lequel vous pouvez télécharger des photos. L'application vous permet de supprimer les photos et les albums que vous ajoutez.



Le script de navigateur utilise le SDK JavaScript pour interagir avec un compartiment Amazon S3. Utilisez les méthodes suivantes de la classe client Amazon S3 pour activer l'application d'album photo :

- `listObjects`
- `headObject`
- `putObject`
- `upload`
- `deleteObject`
- `deleteObjects`

Tâches prérequises

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

- Dans la console Amazon S3, créez un compartiment Amazon S3 que vous utiliserez pour stocker les photos de l'album. Pour plus d'informations sur la création d'un bucket dans la console, consultez Creating a Bucket dans le guide de l'utilisateur d'Amazon Simple Storage Service. Veillez à disposer des autorisations Read (Lecture) et Write (Écriture) sur Objets. Pour plus d'informations sur la définition des autorisations de bucket, consultez la section Configuration des autorisations pour l'accès au site Web.
- Dans la console Amazon Cognito, créez un pool d'identités Amazon Cognito à l'aide d'identités fédérées avec un accès activé pour les utilisateurs non authentifiés de la même région que le compartiment Amazon S3. Vous devez inclure l'ID du groupe d'identités dans le code afin d'obtenir

les informations d'identification relatives au script du navigateur. Pour plus d'informations sur les identités fédérées Amazon Cognito, consultez les groupes d'identités Amazon Cognito (identités fédérées) dans le guide du développeur Amazon Cognito.

- Dans la console IAM, recherchez le rôle IAM créé par Amazon Cognito pour les utilisateurs non authentifiés. Ajoutez la politique suivante pour accorder des autorisations de lecture et d'écriture à un compartiment Amazon S3. Pour plus d'informations sur la création d'un rôle IAM, consultez la section Création d'un rôle pour déléguer des autorisations à un AWS service dans le Guide de l'utilisateur IAM.

  Utilisez cette politique de rôle pour le rôle IAM créé par Amazon Cognito pour les utilisateurs non authentifiés.

  > **Warning**
  >
  > Si vous activez l'accès pour des utilisateurs non authentifiés, vous accordez un accès total en écriture sur le compartiment et tous les objets du compartiment. Dans cet exemple, cette démarche liée à la sécurité est utile pour mettre l'accent sur les principaux objectifs. Cependant, dans de nombreuses situations réelles, une sécurité plus stricte, comme l'utilisation d'utilisateurs authentifiés et la propriété d'objet, est fortement recommandée.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:DeleteObject",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:PutObject",
            "s3:PutObjectAcl"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME",
            "arn:aws:s3:::BUCKET_NAME/*"
         ]
      }
   ]
```

```
}
```

Configuration de CORS

Avant que le script du navigateur puisse accéder au compartiment Amazon S3, vous devez d'abord configurer sa configuration CORS comme suit.

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

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "ETag"
        ]
    }
]
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <CORSRule>
```

```
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>POST</AllowedMethod>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>PUT</AllowedMethod>
        <AllowedMethod>DELETE</AllowedMethod>
        <AllowedMethod>HEAD</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
        <ExposeHeader>ETag</ExposeHeader>
    </CORSRule>
</CORSConfiguration>
```

La page web

Le code HTML pour l'application de chargement de photos se compose d'un élément ** ** dans lequel le script de navigateur crée l'interface utilisateur de chargement. Le premier élément **<script>** ajoute le kit SDK au script de navigateur. Le deuxième **<script>**élément ajoute le JavaScript fichier externe qui contient le code du script du navigateur.

```
<!DOCTYPE html>
<html>
  <head>
     <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./s3_photoExample.js"></script>
    <script>
       function getHtml(template) {
          return template.join('\n');
       }
       listAlbums();
    </script>
  </head>
  <body>
    <h1>My Photo Albums App</h1>
    <div id="app"></div>
  </body>
</html>
```

Configuration du kit SDK

Obtenez les informations d'identification nécessaires pour configurer le SDK en appelant la `CognitoIdentityCredentials` méthode, en fournissant l'ID du pool d'identités Amazon Cognito. Ensuite, créez un objet de service `AWS.S3`.

```
var albumBucketName = "BUCKET_NAME";
var bucketRegion = "REGION";
var IdentityPoolId = "IDENTITY_POOL_ID";

AWS.config.update({
  region: bucketRegion,
  credentials: new AWS.CognitoIdentityCredentials({
    IdentityPoolId: IdentityPoolId,
  }),
});

var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});
```

Dans cet exemple, presque tout le code restant est organisé en une série de fonctions qui rassemblent et présentent des informations sur les albums dans le compartiment, chargent et affichent les photos téléchargées dans les albums, et suppriment les photos et les albums. Ces fonctions sont :

- `listAlbums`
- `createAlbum`
- `viewAlbum`
- `addPhoto`
- `deleteAlbum`
- `deletePhoto`

Liste des albums dans le compartiment

L'application crée des albums dans le compartiment Amazon S3 sous forme d'objets dont les touches commencent par une barre oblique, indiquant que l'objet fonctionne comme un dossier. Pour

répertorier tous les albums existants, la fonction `listAlbums` de l'application appelle la méthode `listObjects` de l'objet de service `AWS.S3` tout en utilisant `commonPrefix`, afin que l'appel retourne uniquement les objets utilisés en tant qu'albums.

Le reste de la fonction prend la liste des albums du compartiment Amazon S3 et génère le code HTML nécessaire pour afficher la liste des albums sur la page Web. Cela permet également de supprimer et d'ouvrir des albums individuels.

```
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          "<span onclick=\"deleteAlbum('" + albumName + "')\">X</span>",
          "<span onclick=\"viewAlbum('" + albumName + "')\">",
          albumName,
          "</span>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml([
            "<p>Click on an album name to view it.</p>",
            "<p>Click on the X to delete the album.</p>",
          ])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
        "<button onclick=\"createAlbum(prompt('Enter Album Name:'))\">",
        "Create New Album",
        "</button>",
      ];
      document.getElementById("app").innerHTML = getHtml(htmlTemplate);
    }
```

```
  });
}
```

Création d'un album dans le compartiment

Pour créer un album dans le compartiment Amazon S3, la `createAlbum` fonction de l'application valide d'abord le nom donné au nouvel album afin de s'assurer qu'il contient les caractères appropriés. La fonction forme ensuite une clé d'objet Amazon S3, qu'elle transmet à la `headObject` méthode de l'objet de service Amazon S3. Cette méthode retourne les métadonnées pour la clé spécifiée, si bien que si elle retourne des données, cela signifie qu'un objet avec cette clé existe déjà.

Si l'album n'existe pas déjà, la fonction appelle la méthode `putObject` de l'objet de service `AWS.S3` pour créer l'album. Ensuite, elle appelle la fonction `viewAlbum` pour afficher le nouvel album vide.

```
function createAlbum(albumName) {
  albumName = albumName.trim();
  if (!albumName) {
    return alert("Album names must contain at least one non-space character.");
  }
  if (albumName.indexOf("/") !== -1) {
    return alert("Album names cannot contain slashes.");
  }
  var albumKey = encodeURIComponent(albumName);
  s3.headObject({ Key: albumKey }, function (err, data) {
    if (!err) {
      return alert("Album already exists.");
    }
    if (err.code !== "NotFound") {
      return alert("There was an error creating your album: " + err.message);
    }
    s3.putObject({ Key: albumKey }, function (err, data) {
      if (err) {
        return alert("There was an error creating your album: " + err.message);
      }
      alert("Successfully created album.");
      viewAlbum(albumName);
    });
  });
}
```

Affichage d'un album

Pour afficher le contenu d'un album dans le compartiment Amazon S3, la `viewAlbum` fonction de l'application prend un nom d'album et crée la clé Amazon S3 pour cet album. Ensuite, la fonction appelle la méthode `listObjects` de l'objet de service `AWS.S3` pour obtenir une liste de tous les objets (photos) de l'album.

Le reste de la fonction prend la liste des objets (photos) de l'album et génère le code HTML nécessaire à l'affichage des photos dans la page web. Cela permet également de supprimer des photos individuelles et de revenir à la liste de l'album.

```
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Response instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span onclick=\"deletePhoto('" +
          albumName +
          "','" +
          photoKey +
          "')\">",
        "X",
        "</span>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
```

```
    });
    var message = photos.length
      ? "<p>Click on the X to delete the photo</p>"
      : "<p>You do not have any photos in this album. Please add photos.</p>";
    var htmlTemplate = [
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      '<input id="photoupload" type="file" accept="image/*">',
      '<button id="addphoto" onclick="addPhoto(\'' + albumName + "')\">",
      "Add Photo",
      "</button>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
    ];
    document.getElementById("app").innerHTML = getHtml(htmlTemplate);
  });
}
```

Ajout de photos à un album

Pour télécharger une photo dans un album du compartiment Amazon S3, la `addPhoto` fonction de l'application utilise un élément de sélection de fichiers dans la page Web pour identifier le fichier à télécharger. Ensuite, elle forme une clé pour la photo à charger à partir du nom de l'album et du nom du fichier actuels.

La fonction appelle la `upload` méthode de l'objet de service Amazon S3 pour télécharger la photo. Après avoir chargé la photo, la fonction affiche de nouveau l'album afin que la photo chargée apparaisse.

```
function addPhoto(albumName) {
  var files = document.getElementById("photoupload").files;
  if (!files.length) {
    return alert("Please choose a file to upload first.");
  }
  var file = files[0];
  var fileName = file.name;
```

```
  var albumPhotosKey = encodeURIComponent(albumName) + "/";

  var photoKey = albumPhotosKey + fileName;

  // Use S3 ManagedUpload class as it supports multipart uploads
  var upload = new AWS.S3.ManagedUpload({
    params: {
      Bucket: albumBucketName,
      Key: photoKey,
      Body: file,
    },
  });

  var promise = upload.promise();

  promise.then(
    function (data) {
      alert("Successfully uploaded photo.");
      viewAlbum(albumName);
    },
    function (err) {
      return alert("There was an error uploading your photo: ", err.message);
    }
  );
}
```

Suppression d'une photo

Pour supprimer une photo d'un album du compartiment Amazon S3, la `deletePhoto` fonction de l'application appelle la `deleteObject` méthode de l'objet de service Amazon S3. Cela supprime la photo spécifiée par la valeur `photoKey` transmise à la fonction.

```
function deletePhoto(albumName, photoKey) {
  s3.deleteObject({ Key: photoKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your photo: ", err.message);
    }
    alert("Successfully deleted photo.");
    viewAlbum(albumName);
  });
}
```

Suppression d'un album

Pour supprimer un album dans le compartiment Amazon S3, la `deleteAlbum` fonction de l'application appelle la `deleteObjects` méthode de l'objet de service Amazon S3.

```
function deleteAlbum(albumName) {
  var albumKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your album: ", err.message);
    }
    var objects = data.Contents.map(function (object) {
      return { Key: object.Key };
    });
    s3.deleteObjects(
      {
        Delete: { Objects: objects, Quiet: true },
      },
      function (err, data) {
        if (err) {
          return alert("There was an error deleting your album: ", err.message);
        }
        alert("Successfully deleted album.");
        listAlbums();
      }
    );
  });
}
```

Téléchargement de photos sur Amazon S3 : code complet

Cette section contient le code HTML complet et le JavaScript code de l'exemple dans lequel les photos sont chargées dans un album photo Amazon S3. Consultez la section parente pour plus de détails et les conditions préalables.

Code HTML pour l'exemple :

```
<!DOCTYPE html>
<html>
  <head>
     <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
```

```
    <script src="./s3_photoExample.js"></script>
    <script>
       function getHtml(template) {
          return template.join('\n');
       }
       listAlbums();
    </script>
  </head>
  <body>
    <h1>My Photo Albums App</h1>
    <div id="app"></div>
  </body>
</html>
```

Cet exemple de code se trouve ici sur GitHub.

Code de script de navigateur pour l'exemple :

```
var albumBucketName = "BUCKET_NAME";
var bucketRegion = "REGION";
var IdentityPoolId = "IDENTITY_POOL_ID";

AWS.config.update({
  region: bucketRegion,
  credentials: new AWS.CognitoIdentityCredentials({
    IdentityPoolId: IdentityPoolId,
  }),
});

var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
```

```
          "<span onclick=\"deleteAlbum('" + albumName + "')\">X</span>",
          "<span onclick=\"viewAlbum('" + albumName + "')\">",
          albumName,
          "</span>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml([
            "<p>Click on an album name to view it.</p>",
            "<p>Click on the X to delete the album.</p>",
          ])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
        "<button onclick=\"createAlbum(prompt('Enter Album Name:'))\">",
        "Create New Album",
        "</button>",
      ];
      document.getElementById("app").innerHTML = getHtml(htmlTemplate);
    }
  });
}

function createAlbum(albumName) {
  albumName = albumName.trim();
  if (!albumName) {
    return alert("Album names must contain at least one non-space character.");
  }
  if (albumName.indexOf("/") !== -1) {
    return alert("Album names cannot contain slashes.");
  }
  var albumKey = encodeURIComponent(albumName);
  s3.headObject({ Key: albumKey }, function (err, data) {
    if (!err) {
      return alert("Album already exists.");
    }
    if (err.code !== "NotFound") {
      return alert("There was an error creating your album: " + err.message);
    }
```

```
    s3.putObject({ Key: albumKey }, function (err, data) {
      if (err) {
        return alert("There was an error creating your album: " + err.message);
      }
      alert("Successfully created album.");
      viewAlbum(albumName);
    });
  });
}

function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Response instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span onclick=\"deletePhoto('" +
          albumName +
          "','" +
          photoKey +
          "')\">",
        "X",
        "</span>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
```

```
      ? "<p>Click on the X to delete the photo</p>"
      : "<p>You do not have any photos in this album. Please add photos.</p>";
    var htmlTemplate = [
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      '<input id="photoupload" type="file" accept="image/*">',
      '<button id="addphoto" onclick="addPhoto(\'' + albumName + "')\">",
      "Add Photo",
      "</button>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
    ];
    document.getElementById("app").innerHTML = getHtml(htmlTemplate);
  });
}

function addPhoto(albumName) {
  var files = document.getElementById("photoupload").files;
  if (!files.length) {
    return alert("Please choose a file to upload first.");
  }
  var file = files[0];
  var fileName = file.name;
  var albumPhotosKey = encodeURIComponent(albumName) + "/";

  var photoKey = albumPhotosKey + fileName;

  // Use S3 ManagedUpload class as it supports multipart uploads
  var upload = new AWS.S3.ManagedUpload({
    params: {
      Bucket: albumBucketName,
      Key: photoKey,
      Body: file,
    },
  });

  var promise = upload.promise();
```

```
  promise.then(
    function (data) {
      alert("Successfully uploaded photo.");
      viewAlbum(albumName);
    },
    function (err) {
      return alert("There was an error uploading your photo: ", err.message);
    }
  );
}

function deletePhoto(albumName, photoKey) {
  s3.deleteObject({ Key: photoKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your photo: ", err.message);
    }
    alert("Successfully deleted photo.");
    viewAlbum(albumName);
  });
}

function deleteAlbum(albumName) {
  var albumKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your album: ", err.message);
    }
    var objects = data.Contents.map(function (object) {
      return { Key: object.Key };
    });
    s3.deleteObjects(
      {
        Delete: { Objects: objects, Quiet: true },
      },
      function (err, data) {
        if (err) {
          return alert("There was an error deleting your album: ", err.message);
        }
        alert("Successfully deleted album.");
        listAlbums();
      }
    );
  });
```

```
}
```

Cet exemple de code se trouve ici sur GitHub.

Exemples Node.js d'Amazon S3

Les rubriques suivantes présentent des exemples de la manière dont ils AWS SDK for JavaScript peuvent être utilisés pour interagir avec les compartiments Amazon S3 à l'aide de Node.js.

Rubriques

- Création et utilisation de compartiments Amazon S3
- Configuration de compartiments Amazon S3
- Gestion des autorisations d'accès aux compartiments Amazon S3
- Utilisation de stratégies de compartiment Amazon S3
- Utilisation d'un compartiment Amazon S3 en tant qu'hôte Web statique

Création et utilisation de compartiments Amazon S3

Cet exemple de code Node.js présente :

- Comment obtenir et afficher une liste des compartiments Amazon S3 dans votre compte.
- Comment créer un compartiment Amazon S3.
- Comment charger un objet dans un compartiment spécifié.

Scénario

Dans cet exemple, une série de modules Node.js sont utilisés pour obtenir une liste des compartiments Amazon S3 existants, créer un compartiment et charger un fichier dans un compartiment spécifié. Ces modules Node.js utilisent le SDK pour JavaScript obtenir des informations et télécharger des fichiers dans un compartiment Amazon S3 en utilisant les méthodes suivantes de la classe client Amazon S3 :

- `listBuckets`
- `createBucket`
- `listObjects`
- `upload`
- `deleteBucket`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Configuration du kit SDK

Configurez le SDK pour JavaScript en créant un objet de configuration global, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Afficher la liste des compartiments Amazon S3

Créez un module Node.js nommé `s3_listbuckets.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon Simple Storage Service, créez un objet `AWS.S3` de service. Appelez la `listBuckets` méthode de l'objet de service Amazon S3 pour récupérer la liste de vos buckets. Le paramètre `data` de la fonction de rappel dispose d'une propriété `Buckets` contenant un tableau de cartes pour représenter les compartiments. Affichez la liste des compartiments en la connectant à la console.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Call S3 to list the buckets
s3.listBuckets(function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Buckets);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_listbuckets.js
```

Cet exemple de code se trouve ici sur GitHub.

Création d’un compartiment Amazon S3

Créez un module Node.js nommé `s3_createbucket.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Créez un objet de service `AWS.S3`. Le module prend un seul argument de ligne de commande pour spécifier un nom pour le nouveau compartiment.

Ajoutez une variable contenant les paramètres utilisés pour appeler la `createBucket` méthode de l'objet de service Amazon S3, y compris le nom du compartiment nouvellement créé. La fonction de rappel enregistre l'emplacement du nouveau compartiment sur la console une fois qu'Amazon S3 l'a créé avec succès.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });
```

```
// Create the parameters for calling createBucket
var bucketParams = {
  Bucket: process.argv[2],
};

// call S3 to create the bucket
s3.createBucket(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Location);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_createbucket.js BUCKET_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Chargement d'un fichier dans un compartiment Amazon S3

Créez un module Node.js nommé `s3_upload.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Créez un objet de service `AWS.S3`. Le module prend deux arguments de ligne de commande : le premier pour spécifier le compartiment de destination et le deuxième pour spécifier le fichier à charger.

Créez une variable avec les paramètres nécessaires pour appeler la `upload` méthode de l'objet de service Amazon S3. Indiquez le nom du compartiment cible dans le paramètre `Bucket`. Le paramètre `Key` est défini sur le nom du fichier sélectionné que vous pouvez obtenir à l'aide du module `path` Node.js. Le paramètre `Body` est défini sur les contenus du fichier que vous pouvez obtenir à l'aide de `createReadStream` à partir du module `fs` Node.js.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
var s3 = new AWS.S3({ apiVersion: "2006-03-01" });
```

```
// call S3 to retrieve upload file to specified bucket
var uploadParams = { Bucket: process.argv[2], Key: "", Body: "" };
var file = process.argv[3];

// Configure the file stream and obtain the upload parameters
var fs = require("fs");
var fileStream = fs.createReadStream(file);
fileStream.on("error", function (err) {
  console.log("File Error", err);
});
uploadParams.Body = fileStream;
var path = require("path");
uploadParams.Key = path.basename(file);

// call S3 to retrieve upload file to specified bucket
s3.upload(uploadParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  }
  if (data) {
    console.log("Upload Success", data.Location);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_upload.js BUCKET_NAME FILE_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Répertorier des objets dans un compartiment Amazon S3

Créez un module Node.js nommé `s3_listobjects.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Créez un objet de service `AWS.S3`.

Ajoutez une variable contenant les paramètres utilisés pour appeler la `listObjects` méthode de l'objet de service Amazon S3, y compris le nom du compartiment à lire. La fonction de rappel journalise une liste d'objets (fichiers) ou un message d'erreur.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create the parameters for calling listObjects
var bucketParams = {
  Bucket: "BUCKET_NAME",
};

// Call S3 to obtain a list of the objects in the bucket
s3.listObjects(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_listobjects.js
```

Cet exemple de code se trouve ici sur GitHub.

Supprimer un compartiment Amazon S3

Créez un module Node.js nommé `s3_deletebucket.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Créez un objet de service `AWS.S3`.

Ajoutez une variable contenant les paramètres utilisés pour appeler la `createBucket` méthode de l'objet de service Amazon S3, y compris le nom du compartiment à supprimer. Pour être supprimé, le compartiment doit être vide. La fonction de rappel journalise un message de réussite ou d'échec.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });
```

```
// Create params for S3.deleteBucket
var bucketParams = {
  Bucket: "BUCKET_NAME",
};

// Call S3 to delete the bucket
s3.deleteBucket(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_deletebucket.js
```

Cet exemple de code se trouve ici sur GitHub.

Configuration de compartiments Amazon S3

Cet exemple de code Node.js présente :

- Comment configurer les autorisations de partage des ressources cross-origine (CORS) d'un compartiment.

Scénario

Dans cet exemple, une série de modules Node.js est utilisée pour répertorier vos compartiments Amazon S3 et configurer le partage CORS ainsi que la journalisation des compartiments. Les modules Node.js utilisent le SDK pour configurer un compartiment Amazon S3 sélectionné JavaScript à l'aide des méthodes suivantes de la classe client Amazon S3 :

- `getBucketCors`

- `putBucketCors`

Pour plus d'informations sur l'utilisation de la configuration CORS avec un compartiment Amazon S3, consultez Cross-Origin Resource Sharing (CORS) dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Configuration du kit SDK

Configurez le SDK pour JavaScript en créant un objet de configuration global, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Extraction de la configuration CORS d'un compartiment

Créez un module Node.js nommé `s3_getcors.js`. Le module prend un seul argument de ligne de commande pour spécifier le compartiment de la configuration CORS que vous souhaitez. Veillez à configurer le kit SDK comme indiqué précédemment. Créez un objet de service `AWS.S3`.

Le seul paramètre à passer est le nom du compartiment sélectionné lorsque vous appelez la méthode `getBucketCors`. Si le compartiment possède actuellement une configuration CORS, cette configuration est renvoyée par Amazon S3 en tant que `CORSRules` propriété du `data` paramètre transmis à la fonction de rappel.

Si le compartiment sélectionné ne dispose pas de configuration CORS, ces informations sont retournées à la fonction de rappel dans le paramètre `error`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Set the parameters for S3.getBucketCors
var bucketParams = { Bucket: process.argv[2] };

// call S3 to retrieve CORS configuration for selected bucket
s3.getBucketCors(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", JSON.stringify(data.CORSRules));
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_getcors.js BUCKET_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Définition d'une configuration CORS de compartiment

Créez un module Node.js nommé `s3_setcors.js`. Le module prend plusieurs arguments de ligne de commande: le premier spécifie le compartiment de la configuration CORS que vous souhaitez définir. D'autres arguments énumèrent les méthodes HTTP (POST, GET, PUT, PATCH, DELETE, POST) que vous souhaitez autoriser pour le compartiment. Configurez le kit SDK comme illustré précédemment.

Créez un objet de service `AWS.S3`. Ensuite, créez un objet JSON pour contenir les valeur de la configuration CORS selon les besoins de la méthode `putBucketCors` de l'objet de service `AWS.S3`. Spécifiez `"Authorization"` pour la valeur `AllowedHeaders` et `"*"` pour la valeur `AllowedOrigins`. Tout d'abord, définissez la valeur `AllowedMethods` comme un tableau vide.

Spécifiez les méthodes autorisées comme paramètres de ligne de commande pour le module Node.js, en ajoutant chacune des méthodes correspondant à l'un des paramètres. Ajoutez la

configuration CORS obtenue au tableau des configurations du paramètre `CORSRules`. Spécifiez le compartiment que vous souhaitez configurer pour CORS dans le paramètre `Bucket`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create initial parameters JSON for putBucketCors
var thisConfig = {
  AllowedHeaders: ["Authorization"],
  AllowedMethods: [],
  AllowedOrigins: ["*"],
  ExposeHeaders: [],
  MaxAgeSeconds: 3000,
};

// Assemble the list of allowed methods based on command line parameters
var allowedMethods = [];
process.argv.forEach(function (val, index, array) {
  if (val.toUpperCase() === "POST") {
    allowedMethods.push("POST");
  }
  if (val.toUpperCase() === "GET") {
    allowedMethods.push("GET");
  }
  if (val.toUpperCase() === "PUT") {
    allowedMethods.push("PUT");
  }
  if (val.toUpperCase() === "PATCH") {
    allowedMethods.push("PATCH");
  }
  if (val.toUpperCase() === "DELETE") {
    allowedMethods.push("DELETE");
  }
  if (val.toUpperCase() === "HEAD") {
    allowedMethods.push("HEAD");
  }
});
```

```
// Copy the array of allowed methods into the config object
thisConfig.AllowedMethods = allowedMethods;
// Create array of configs then add the config object to it
var corsRules = new Array(thisConfig);

// Create CORS params
var corsParams = {
  Bucket: process.argv[2],
  CORSConfiguration: { CORSRules: corsRules },
};

// set the new CORS configuration on the selected bucket
s3.putBucketCors(corsParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    // update the displayed CORS config for the selected bucket
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, saisissez la ligne de commande suivante, y compris une ou plusieurs méthodes HTTP comme illustré.

```
node s3_setcors.js BUCKET_NAME get put
```

Cet exemple de code se trouve ici sur GitHub.

Gestion des autorisations d’accès aux compartiments Amazon S3

Cet exemple de code Node.js présente :

- Comment extraire ou définir la liste de contrôle d'accès pour un compartiment Amazon S3.

Scénario

Dans cet exemple, un module Node.js est utilisé pour afficher la liste de contrôle d'accès (ACL) de compartiment d'un compartiment sélectionné et appliquer les modifications à la liste de contrôle d'accès (ACL) pour un compartiment sélectionné. Le module Node.js utilise le SDK pour gérer les autorisations JavaScript d'accès aux compartiments Amazon S3 à l'aide des méthodes suivantes de la classe client Amazon S3 :

- `getBucketAcl`
- `putBucketAcl`

Pour plus d'informations sur les listes de contrôle d'accès pour les compartiments Amazon S3, consultez la section Gestion de l'accès avec des ACL dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Configuration du kit SDK

Configurez le SDK pour JavaScript en créant un objet de configuration global, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Extraction de la liste de contrôle d'accès du compartiment actuel

Créez un module Node.js nommé `s3_getbucketacl.js`. Le module prend un seul argument de ligne de commande pour spécifier le compartiment de la configuration ACL que vous souhaitez. Veillez à configurer le kit SDK comme indiqué précédemment.

Créez un objet de service `AWS.S3`. Le seul paramètre à passer est le nom du compartiment sélectionné lorsque vous appelez la méthode `getBucketAcl`. La configuration actuelle de la liste de contrôle d'accès est renvoyée par Amazon S3 dans le `data` paramètre transmis à la fonction de rappel.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to retrieve policy for selected bucket
s3.getBucketAcl(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data.Grants);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_getbucketacl.js BUCKET_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation de stratégies de compartiment Amazon S3

Cet exemple de code Node.js présente :

- Comment récupérer la politique de compartiment d'un compartiment Amazon S3.
- Comment ajouter ou mettre à jour la politique de compartiment d'un compartiment Amazon S3.
- Comment supprimer la politique de compartiment d'un compartiment Amazon S3.

Scénario

Dans cet exemple, une série de modules Node.js sont utilisés pour récupérer, définir ou supprimer une politique de compartiment sur un compartiment Amazon S3. Les modules Node.js utilisent le SDK JavaScript pour configurer la politique d'un compartiment Amazon S3 sélectionné à l'aide des méthodes suivantes de la classe client Amazon S3 :

- `getBucketPolicy`
- `putBucketPolicy`
- `deleteBucketPolicy`

Pour plus d'informations sur les politiques de compartiment pour les compartiments Amazon S3, consultez la section Utilisation des politiques de compartiment et des politiques utilisateur dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Configuration du kit SDK

Configurez le SDK pour JavaScript en créant un objet de configuration global, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`.

```
// Load the SDK for JavaScript
```

```
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Extraction de la stratégie de compartiment actuelle

Créez un module Node.js nommé `s3_getbucketpolicy.js`. Le module prend un seul argument de ligne de commande pour spécifier le compartiment de la stratégie que vous souhaitez. Veillez à configurer le kit SDK comme indiqué précédemment.

Créez un objet de service `AWS.S3`. Le seul paramètre à passer est le nom du compartiment sélectionné lorsque vous appelez la méthode `getBucketPolicy`. Si le compartiment possède actuellement une politique, cette politique est renvoyée par Amazon S3 dans le `data` paramètre transmis à la fonction de rappel.

Si le compartiment sélectionné ne dispose pas de stratégie, ces informations sont retournées à la fonction de rappel dans le paramètre `error`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to retrieve policy for selected bucket
s3.getBucketPolicy(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data.Policy);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_getbucketpolicy.js BUCKET_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Définition d'une stratégie de compartiment simple

Créez un module Node.js nommé `s3_setbucketpolicy.js`. Le module prend un seul argument de ligne de commande pour spécifier le compartiment de la stratégie que vous souhaitez appliquer. Configurez le kit SDK comme illustré précédemment.

Créez un objet de service `AWS.S3`. Les stratégies de compartiment sont spécifiées au format JSON. Tout d'abord, créez un objet JSON contenant toutes les valeurs pour spécifier la stratégie à l'exception de la valeur `Resource` qui identifie le compartiment.

Formatez la chaîne `Resource` requise par la stratégie, en intégrant le nom du compartiment sélectionné. Insérez cette chaîne dans l'objet JSON. Préparez les paramètres pour la méthode `putBucketPolicy`, y compris le nom du compartiment et la stratégie JSON convertie en une valeur de chaîne.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var readOnlyAnonUserPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "AddPerm",
      Effect: "Allow",
      Principal: "*",
      Action: ["s3:GetObject"],
      Resource: [""],
    },
  ],
};

// create selected bucket resource string for bucket policy
var bucketResource = "arn:aws:s3:::" + process.argv[2] + "/*";
readOnlyAnonUserPolicy.Statement[0].Resource[0] = bucketResource;

// convert policy JSON into string and assign into params
var bucketPolicyParams = {
```

```
  Bucket: process.argv[2],
  Policy: JSON.stringify(readOnlyAnonUserPolicy),
};

// set the new policy on the selected bucket
s3.putBucketPolicy(bucketPolicyParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_setbucketpolicy.js BUCKET_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'une stratégie de compartiment

Créez un module Node.js nommé `s3_deletebucketpolicy.js`. Le module prend un seul argument de ligne de commande pour spécifier le compartiment de la stratégie que vous souhaitez supprimer. Configurez le kit SDK comme illustré précédemment.

Créez un objet de service `AWS.S3`. Le seul paramètre à passer lorsque vous appelez la méthode `deleteBucketPolicy` est le nom du compartiment sélectionné.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to delete policy for selected bucket
s3.deleteBucketPolicy(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else if (data) {
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_deletebucketpolicy.js BUCKET_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation d'un compartiment Amazon S3 en tant qu'hôte Web statique

Cet exemple de code Node.js présente :

- Comment configurer un compartiment Amazon S3 en tant qu'hôte Web statique.

Scénario

Dans cet exemple, une série de modules Node.js est utilisée pour configurer tous vos compartiments comme hôte web statique. Les modules Node.js utilisent le SDK pour configurer un compartiment Amazon S3 sélectionné JavaScript à l'aide des méthodes suivantes de la classe client Amazon S3 :

- `getBucketWebsite`
- `putBucketWebsite`
- `deleteBucketWebsite`

Pour plus d'informations sur l'utilisation d'un compartiment Amazon S3 en tant qu'hôte Web statique, consultez la section Hébergement d'un site Web statique sur Amazon S3 dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Configuration du kit SDK

Configurez le SDK pour JavaScript en créant un objet de configuration global, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Extraction de la configuration actuelle du site web d'un compartiment

Créez un module Node.js nommé `s3_getbucketwebsite.js`. Le module prend un seul argument de ligne de commande pour spécifier le compartiment de la configuration de site web que vous souhaitez. Configurez le kit SDK comme illustré précédemment.

Créez un objet de service `AWS.S3`. Créez une fonction qui extraie la configuration de site web du compartiment actuel pour le compartiment sélectionné dans la liste de compartiments. Le seul paramètre à passer est le nom du compartiment sélectionné lorsque vous appelez la méthode `getBucketWebsite`. Si le bucket possède actuellement une configuration de site Web, cette configuration est renvoyée par Amazon S3 dans le `data` paramètre transmis à la fonction de rappel.

Si le compartiment sélectionné ne dispose pas de configuration de site web, ces informations sont retournées à la fonction de rappel dans le paramètre `err`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
```

```
// call S3 to retrieve the website configuration for selected bucket
s3.getBucketWebsite(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_getbucketwebsite.js BUCKET_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Définition de la configuration de site web d'un compartiment

Créez un module Node.js nommé `s3_setbucketwebsite.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Créez un objet de service `AWS.S3`.

Créez une fonction qui applique une configuration de site web d'un compartiment. La configuration permet au compartiment sélectionné d'être utilisé comme hôte web statique. Les configurations de site web sont spécifiées au format JSON. Tout d'abord, créez un objet JSON contenant toutes les valeurs pour spécifier la configuration de site web, à l'exception de la valeur `Key` qui identifie le document d'erreur et la valeur `Suffix` qui identifie le document d'index.

Insérez les valeurs des éléments d'entrée de texte dans l'objet JSON. Préparez les paramètres pour la méthode `putBucketWebsite`, y compris le nom du compartiment et la configuration de site web JSON.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create JSON for putBucketWebsite parameters
var staticHostParams = {
  Bucket: "",
```

```
  WebsiteConfiguration: {
    ErrorDocument: {
      Key: "",
    },
    IndexDocument: {
      Suffix: "",
    },
  },
};

// Insert specified bucket name and index and error documents into params JSON
// from command line arguments
staticHostParams.Bucket = process.argv[2];
staticHostParams.WebsiteConfiguration.IndexDocument.Suffix = process.argv[3];
staticHostParams.WebsiteConfiguration.ErrorDocument.Key = process.argv[4];

// set the new website configuration on the selected bucket
s3.putBucketWebsite(staticHostParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    // update the displayed website configuration for the selected bucket
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_setbucketwebsite.js BUCKET_NAME INDEX_PAGE ERROR_PAGE
```

Cet exemple de code se trouve ici sur GitHub.

Suppression de la configuration de site web d'un compartiment

Créez un module Node.js nommé `s3_deletebucketwebsite.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Créez un objet de service `AWS.S3`.

Créez une fonction qui supprime la configuration de site web du compartiment sélectionné. Le seul paramètre à passer lorsque vous appelez la méthode `deleteBucketWebsite` est le nom du compartiment sélectionné.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };

// call S3 to delete website configuration for selected bucket
s3.deleteBucketWebsite(bucketParams, function (error, data) {
  if (error) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node s3_deletebucketwebsite.js BUCKET_NAME
```

Cet exemple de code se trouve ici sur GitHub.

Exemples de services de messagerie Amazon Simple

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 `AWS.SES` client. Pour plus d'informations sur l'utilisation de la classe client Amazon SES, consultez `Class: AWS.SES`la référence de l'API.

Rubriques

- Gestion des identités Amazon SES
- Utilisation de modèles d'e-mails dans Amazon SES
- Envoyer un e-mail à l'aide d'Amazon SES
- Utilisation de filtres d'adresses IP pour la réception d'e-mails dans Amazon SES
- Utilisation des règles de réception dans 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 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'envoi d'autorisations dans Amazon SES, consultez Présentation de l'autorisation d'envoi Amazon SES.

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 `AWS.SES` client :

- `listIdentities`
- `deleteIdentity`
- `verifyEmailIdentity`
- `verifyDomainIdentity`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier JSON d'informations d'identification, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Configuration du kit SDK

Configurez le SDK pour JavaScript en créant un objet de configuration global, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');

// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Liste de 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 module Node.js nommé `ses_listidentities.js`. Configurez le kit SDK comme illustré précédemment.

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

Traitez ensuite l'élément `response` dans le rappel de promesse. Les `data` retournées par la promesse contiennent un tableau des identités de domaine comme spécifié par le paramètre `IdentityType`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create listIdentities params
var params = {
  IdentityType: "Domain",
  MaxItems: 10,
};

// Create the promise and SES service object
var listIDsPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listIdentities(params)
  .promise();

// Handle promise's fulfilled/rejected states
listIDsPromise
  .then(function (data) {
    console.log(data.Identities);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ses_listidentities.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `ses_verifyemailidentity.js`. Configurez le kit SDK comme illustré précédemment. Pour accéder à Amazon SES, créez un objet `AWS.SES` de service.

Créez un objet pour transmettre le paramètre `EmailAddress` ainsi que les autres paramètres de la méthode `verifyEmailIdentity` de la classe client `AWS.SES`. Pour appeler la méthode `verifyEmailIdentity`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SES service object
var verifyEmailPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .verifyEmailIdentity({ EmailAddress: "ADDRESS@DOMAIN.EXT" })
  .promise();

// Handle promise's fulfilled/rejected states
verifyEmailPromise
  .then(function (data) {
    console.log("Email verification initiated");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

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

```
node ses_verifyemailidentity.js
```

Cet exemple de code se trouve ici sur GitHub.

Vérification d'une identité de domaine

Dans cet exemple, utilisez un module Node.js pour vérifier les domaines de messagerie à utiliser avec Amazon SES. Créez un module Node.js nommé `ses_verifydomainidentity.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre le paramètre `Domain` ainsi que les autres paramètres de la méthode `verifyDomainIdentity` de la classe client `AWS.SES`. Pour appeler la `verifyDomainIdentity` méthode, créez une promesse pour appeler un objet de service Amazon SES, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var verifyDomainPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .verifyDomainIdentity({ Domain: "DOMAIN_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
verifyDomainPromise
  .then(function (data) {
    console.log("Verification Token: " + data.VerificationToken);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

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

```
node ses_verifydomainidentity.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression 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 module Node.js nommé `ses_deleteidentity.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre le paramètre `Identity` ainsi que les autres paramètres de la méthode `deleteIdentity` de la classe client `AWS.SES`. Pour appeler la `deleteIdentity` méthode, créez un `request` pour appeler un objet de service Amazon SES en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var deletePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteIdentity({ Identity: "DOMAIN_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
deletePromise
  .then(function (data) {
    console.log("Identity Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node ses_deleteidentity.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation de modèles d'e-mails dans Amazon SES

Cet exemple de code Node.js présente :

- La récupération d'une liste de toutes vos modèles d'e-mail.
- La récupération et la mise à jour des modèles d'e-mail.
- La création et la suppression des modèles d'e-mail.

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-mail dans Amazon Simple Email Service, 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.

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 `AWS.SES` client :

- `listTemplates`
- `createTemplate`
- `getTemplate`
- `deleteTemplate`
- `updateTemplate`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.

- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur la création d'un fichier d'informations d'identification, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Liste de vos modèles 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 module Node.js nommé `ses_listtemplates.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre les paramètres de la méthode `listTemplates` de la classe client `AWS.SES`. Pour appeler la méthode `listTemplates`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listTemplates({ MaxItems: ITEMS_COUNT })
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

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

```
node ses_listtemplates.js
```

Cet exemple de code se trouve ici sur GitHub.

Récupération d'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 module Node.js nommé `ses_gettemplate.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre le paramètre `TemplateName` ainsi que les autres paramètres de la méthode `getTemplate` de la classe client `AWS.SES`. Pour appeler la méthode `getTemplate`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create the promise and Amazon Simple Email Service (Amazon SES) service object.
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .getTemplate({ TemplateName: "TEMPLATE_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log(data.Template.SubjectPart);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

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

```
node ses_gettemplate.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `ses_createtemplate.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre les paramètres de la méthode `createTemplate` de la classe client `AWS.SES`, y compris `TemplateName`, `HtmlPart`, `SubjectPart` et `TextPart`. Pour appeler la méthode `createTemplate`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createTemplate params
var params = {
  Template: {
    TemplateName: "TEMPLATE_NAME" /* required */,
    HtmlPart: "HTML_CONTENT",
    SubjectPart: "SUBJECT_LINE",
    TextPart: "TEXT_CONTENT",
  },
};

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

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

```
node ses_createtemplate.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `ses_updatetemplate.js`. Configurez le kit SDK comme illustré précédemment.

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 `updateTemplate` de la classe client `AWS.SES`. Pour appeler la méthode `updateTemplate`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create updateTemplate parameters
var params = {
  Template: {
    TemplateName: "TEMPLATE_NAME" /* required */,
    HtmlPart: "HTML_CONTENT",
    SubjectPart: "SUBJECT_LINE",
    TextPart: "TEXT_CONTENT",
  },
};

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .updateTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log("Template Updated");
  })
```

```
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

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

```
node ses_updatetemplate.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `ses_deletetemplate.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre le paramètre`TemplateName` obligatoire à la méthode `deleteTemplate` de la classe client `AWS.SES`. Pour appeler la méthode `deleteTemplate`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteTemplate({ TemplateName: "TEMPLATE_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log("Template Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

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

```
node ses_deletetemplate.js
```

Cet exemple de code se trouve ici sur 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 les sections 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.

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 `AWS.SES` client :

- `sendEmail`
- `sendTemplatedEmail`
- `sendBulkTemplatedEmail`

Tâches prérequises

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier JSON d'informations d'identification, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Exigences d'envoi d'un e-mail

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 `SES.sendEmail`, 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 du destinataire n'est pas valide (c'est-à-dire, pas au format `UserName@[SubDomain.]Domain.TopLevelDomain`), le message est rejeté, même si ce dernier contient d'autres destinataires valides.
- Le message ne peut pas comporter plus de 50 destinataires répartis dans les champs À :, Cc : et Cci :. 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 module Node.js nommé `ses_sendemail.js`. Configurez le kit SDK comme illustré précédemment.

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 `sendEmail` de la classe client `AWS.SES`. Pour appeler la méthode `sendEmail`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create sendEmail params
var params = {
  Destination: {
    /* required */
    CcAddresses: [
      "EMAIL_ADDRESS",
      /* more items */
    ],
    ToAddresses: [
      "EMAIL_ADDRESS",
      /* more items */
    ],
  },
  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: "Test email",
    },
  },
  Source: "SENDER_EMAIL_ADDRESS" /* required */,
```

```
  ReplyToAddresses: [
    "EMAIL_ADDRESS",
    /* more items */
  ],
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne 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 sur GitHub.

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

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES. Créez un module Node.js nommé `ses_sendtemplatedemail.js`. Configurez le kit SDK comme illustré précédemment.

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 `sendTemplatedEmail` de la classe client `AWS.SES`. Pour appeler la méthode `sendTemplatedEmail`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create sendTemplatedEmail params
var params = {
  Destination: {
    /* required */
    CcAddresses: [
      "EMAIL_ADDRESS",
      /* more CC email addresses */
    ],
    ToAddresses: [
      "EMAIL_ADDRESS",
      /* more To email addresses */
    ],
  },
  Source: "EMAIL_ADDRESS" /* required */,
  Template: "TEMPLATE_NAME" /* required */,
  TemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }' /* required */,
  ReplyToAddresses: ["EMAIL_ADDRESS"],
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendTemplatedEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne 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 sur GitHub.

Envoi d'un e-mail en bloc 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_sendbulktemplatedemail.js`. Configurez le kit SDK comme illustré précédemment.

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 `sendBulkTemplatedEmail` de la classe client `AWS.SES`. Pour appeler la méthode `sendBulkTemplatedEmail`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create sendBulkTemplatedEmail params
var params = {
  Destinations: [
    /* required */
    {
      Destination: {
        /* required */
        CcAddresses: [
          "EMAIL_ADDRESS",
          /* more items */
        ],
        ToAddresses: [
          "EMAIL_ADDRESS",
          "EMAIL_ADDRESS",
          /* more items */
        ],
      },
      ReplacementTemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }',
    },
  ],
  Source: "EMAIL_ADDRESS" /* required */,
  Template: "TEMPLATE_NAME" /* required */,
  DefaultTemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }',
  ReplyToAddresses: ["EMAIL_ADDRESS"],
};
```

```
// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendBulkTemplatedEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.log(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne 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 sur GitHub.

Utilisation de filtres d'adresses IP pour la réception d'e-mails dans Amazon SES

Cet exemple de code Node.js présente :

- La création de filtres d'adresses IP pour accepter ou rejeter les messages provenant d'une adresse IP ou d'une plage d'adresses IP.
- La liste de vos filtres d'adresses IP actuels.
- La suppression d'un filtre d'adresses IP

Dans Amazon SES, un filtre est une structure de données composée d'un nom, d'une plage d'adresses IP et indiquant s'il faut autoriser ou bloquer le courrier provenant de ce filtre. Les adresses

IP que vous souhaitez bloquer ou autoriser sont spécifiées comme adresse IP unique ou plage d'adresses IP en notation CIDR (Classless Inter-Domain Routing). Pour en savoir plus sur la manière dont Amazon SES reçoit les e-mails, consultez les concepts de réception d'e-mails d'Amazon SES dans le manuel Amazon Simple Email Service Developer Guide.

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 `AWS.SES` client :

- `createReceiptFilter`
- `listReceiptFilters`
- `deleteReceiptFilter`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Configuration du kit SDK

Configurez le SDK pour JavaScript en créant un objet de configuration global, puis en définissant la région pour votre code. Dans cet exemple, la région est `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Création d'un filtre d'adresses IP

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES. Créez un module Node.js nommé `ses_createreceiptfilter.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre les valeurs de paramètre qui définissent le filtre IP, y compris le nom du filtre, une adresse IP ou une plage d'adresses IP à filtrer et l'autorisation ou le blocage du trafic d'e-mails à partir des adresses filtrées. Pour appeler la méthode `createReceiptFilter`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createReceiptFilter params
var params = {
  Filter: {
    IpFilter: {
      Cidr: "IP_ADDRESS_OR_RANGE",
      Policy: "Allow" | "Block",
    },
    Name: "NAME",
  },
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptFilter(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande. Le filtre est créé dans Amazon SES.

```
node ses_createreceiptfilter.js
```

Cet exemple de code se trouve ici sur GitHub.

Liste de vos filtres d'adresses IP.

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES. Créez un module Node.js nommé `ses_listreceiptfilters.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet de paramètre vide. Pour appeler la méthode `listReceiptFilters`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listReceiptFilters({})
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data.Filters);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande. Amazon SES renvoie la liste des filtres.

```
node ses_listreceiptfilters.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un filtre d'adresses IP

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES. Créez un module Node.js nommé `ses_deletereceiptfilter.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre le nom du filtre IP à supprimer. Pour appeler la méthode `deleteReceiptFilter`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptFilter({ FilterName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log("IP Filter deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande. Le filtre est supprimé d'Amazon SES.

```
node ses_deletereceiptfilter.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des règles de réception dans Amazon SES

Cet exemple de code Node.js présente :

- La création et la suppression des règles de réception.
- L'organisation des règles de réception en ensembles de règles de réception.

Les règles de réception d'Amazon SES spécifient ce qu'il faut faire des e-mails reçus pour les adresses e-mail ou les domaines que vous possédez. Une règle de réception contient une condition et une liste ordonnée d'actions. Si le destinataire d'un e-mail entrant correspond à un destinataire spécifié dans les conditions de la règle de réception, Amazon SES exécute les actions spécifiées par la règle de réception.

Pour utiliser Amazon SES comme destinataire d'e-mails, vous devez avoir défini au moins une règle de réception active. Un ensemble de règles de réception est un ensemble ordonné de règles de réception qui spécifient ce qu'Amazon SES doit faire avec le courrier qu'il reçoit sur vos domaines vérifiés. Pour plus d'informations, consultez les sections Création de règles de réception pour la réception d'e-mails d'Amazon SES et Création d'un ensemble de règles de réception pour la réception d'e-mails d'Amazon SES dans le guide du développeur d'Amazon Simple Email Service.

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 `AWS.SES` client :

- `createReceiptRule`
- `deleteReceiptRule`
- `createReceiptRuleSet`
- `deleteReceiptRuleSet`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier JSON d'informations d'identification, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Création d'une règle de réception Amazon S3

Chaque règle de réception pour Amazon SES contient une liste ordonnée d'actions. Cet exemple crée une règle de réception avec une action Amazon S3, qui envoie le message électronique à un compartiment Amazon S3. Pour plus de détails sur les actions relatives aux règles de réception, consultez la section Options d'action du manuel Amazon Simple Email Service Developer Guide.

Pour qu'Amazon SES puisse écrire des e-mails dans un compartiment Amazon S3, créez une politique de compartiment qui `PutObject` autorise Amazon SES. Pour plus d'informations sur la création de cette politique de compartiment, consultez la section Autoriser Amazon SES à écrire dans votre compartiment Amazon S3 dans le manuel Amazon Simple Email Service Developer Guide.

Dans cet exemple, utilisez un module Node.js pour créer une règle de réception dans Amazon SES afin d'enregistrer les messages reçus dans un compartiment Amazon S3. Créez un module Node.js nommé `ses_createreceiptrule.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet de paramètre pour transmettre les valeurs nécessaires à la création de l'ensemble de règles de réception. Pour appeler la méthode `createReceiptRuleSet`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createReceiptRule params
var params = {
  Rule: {
```

```
    Actions: [
      {
        S3Action: {
          BucketName: "S3_BUCKET_NAME",
          ObjectKeyPrefix: "email",
        },
      },
    ],
    Recipients: [
      "DOMAIN | EMAIL_ADDRESS",
      /* more items */
    ],
    Enabled: true | false,
    Name: "RULE_NAME",
    ScanEnabled: true | false,
    TlsPolicy: "Optional",
  },
  RuleSetName: "RULE_SET_NAME",
};

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptRule(params)
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log("Rule created");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande. Amazon SES crée la règle de réception.

```
node ses_createreceiptrule.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'une règle de réception

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES. Créez un module Node.js nommé `ses_deletereceiptrule.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet de paramètre pour transmettre le nom de la règle de réception à supprimer. Pour appeler la méthode `deleteReceiptRule`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create deleteReceiptRule params
var params = {
  RuleName: "RULE_NAME" /* required */,
  RuleSetName: "RULE_SET_NAME" /* required */,
};

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptRule(params)
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log("Receipt Rule Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande. Amazon SES crée la liste des règles de réception définies.

```
node ses_deletereceiptrule.js
```

Cet exemple de code se trouve ici sur GitHub.

Création d'un ensemble de règles de réception

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES. Créez un module Node.js nommé `ses_createreceiptruleset.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet de paramètre pour transmettre le nom du nouvel ensemble de règles de réception. Pour appeler la méthode `createReceiptRuleSet`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptRuleSet({ RuleSetName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande. Amazon SES crée la liste des règles de réception définies.

```
node ses_createreceiptruleset.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'un ensemble de règles de réception

Dans cet exemple, utilisez un module Node.js pour envoyer un e-mail avec Amazon SES. Créez un module Node.js nommé `ses_deletereceiptruleset.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet de paramètre pour transmettre le nom de l'ensemble de règles de réception à supprimer. Pour appeler la méthode `deleeReceiptRuleSet`, créez une promesse pour appeler un objet de service Amazon SES, en transmettant les paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptRuleSet({ RuleSetName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande. Amazon SES crée la liste des règles de réception définies.

```
node ses_deletereceiptruleset.js
```

Cet exemple de code se trouve ici sur GitHub.

Exemples du service de notification Amazon Simple

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, fonctions Lambda) 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 pour Amazon SNS est exposée via le. `Class: AWS.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 `AWS.SNS` client :

- `createTopic`
- `listTopics`
- `deleteTopic`
- `getTopicAttributes`
- `setTopicAttributes`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier JSON d'informations d'identification, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Création d'une rubrique

Dans cet exemple, utilisez un module Node.js pour créer une rubrique Amazon SNS. Créez un module Node.js nommé `sns_createtopic.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet pour transmettre le paramètre`Name` de la nouvelle rubrique à la méthode `createTopic` de la classe client `AWS.SNS`. Pour appeler la `createTopic` méthode, créez une

promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse. Les `data` renvoyées par la promesse contiennent l'ARN de la rubrique.

```
// 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 createTopicPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .createTopic({ Name: "TOPIC_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
createTopicPromise
  .then(function (data) {
    console.log("Topic ARN is " + data.TopicArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_createtopic.js
```

Cet exemple de code se trouve ici sur GitHub.

Liste de vos rubriques

Dans cet exemple, utilisez un module Node.js pour répertorier toutes les rubriques Amazon SNS. Créez un module Node.js nommé `sns_listtopics.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet vide à transmettre à la méthode `listTopics` de la classe client `AWS.SNS`. Pour appeler la `listTopics` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse. Les `data` renvoyées par la promesse contiennent un tableau de vos ARN de rubrique.

```
// 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 listTopicsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listTopics({})
  .promise();

// Handle promise's fulfilled/rejected states
listTopicsPromise
  .then(function (data) {
    console.log(data.Topics);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_listtopics.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 module Node.js nommé `sns_deletetopic.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant le paramètre `TopicArn` de la rubrique à supprimer pour le transmettre à la méthode `deleteTopic` de la classe client `AWS.SNS`. Pour appeler la `deleteTopic` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// 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 deleteTopicPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .deleteTopic({ TopicArn: "TOPIC_ARN" })
  .promise();

// Handle promise's fulfilled/rejected states
deleteTopicPromise
  .then(function (data) {
    console.log("Topic Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_deletetopic.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `sns_gettopicattributes.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 `getTopicAttributes` de la classe client `AWS.SNS`. Pour appeler la `getTopicAttributes` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_gettopicattributes.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `sns_settopicattributes.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 `setTopicAttributes` de la classe client `AWS.SNS`. Pour appeler la `setTopicAttributes` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create setTopicAttributes parameters
var params = {
  AttributeName: "ATTRIBUTE_NAME" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  AttributeValue: "NEW_ATTRIBUTE_VALUE",
};
```

```
// Create promise and SNS service object
var setTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .setTopicAttributes(params)
  .promise();

// Handle promise's fulfilled/rejected states
setTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_settopicattributes.js
```

Cet exemple de code se trouve ici sur 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 `AWS.SNS` client :

- `publish`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier JSON d'informations d'identification, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Publication d'un message sur une rubrique Amazon SNS

Dans cet exemple, utilisez un module Node.js pour publier un message sur une rubrique Amazon SNS. Créez un module Node.js nommé `sns_publishtotopic.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant les paramètres de publication d'un message, notamment le texte du message et l'ARN de la rubrique Amazon SNS. Pour plus d'informations sur les attributs SMS disponibles, consultez SetSMSAttributes.

Transmettez les paramètres à la méthode `publish` de la classe client `AWS.SNS`. Créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément response dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create publish parameters
var params = {
  Message: "MESSAGE_TEXT" /* required */,
  TopicArn: "TOPIC_ARN",
};

// Create promise and SNS service object
var publishTextPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .publish(params)
  .promise();
```

```
// Handle promise's fulfilled/rejected states
publishTextPromise
  .then(function (data) {
    console.log(
      `Message ${params.Message} sent to the topic ${params.TopicArn}`
    );
    console.log("MessageID is " + data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_publishtotopic.js
```

Cet exemple de code se trouve ici sur 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 `AWS.SNS` client :

- `subscribe`

- confirmSubscription
- listSubscriptionsByTopic
- unsubscribe

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier JSON d'informations d'identification, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

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 module Node.js nommé `sns_listsubscriptions.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 `listSubscriptionsByTopic` de la classe client `AWS.SNS`. Pour appeler la `listSubscriptionsByTopic` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

const params = {
  TopicArn: "TOPIC_ARN",
};

// Create promise and SNS service object
var subslistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listSubscriptionsByTopic(params)
  .promise();
```

```
// Handle promise's fulfilled/rejected states
subslistPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_listsubscriptions.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `sns_subscribeemail.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 `subscribe` de la classe client `AWS.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 `subscribe` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create subscribe/email parameters
var params = {
```

```
  Protocol: "EMAIL" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "EMAIL_ADDRESS",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_subscribeemail.js
```

Cet exemple de code se trouve ici sur GitHub.

Abonnement d'un point de terminaison d'application à une rubrique

Dans cet exemple, utilisez un module Node.js pour abonner un point de terminaison d'application mobile afin qu'il reçoive des notifications d'une rubrique Amazon SNS. Créez un module Node.js nommé `sns_subscribeapp.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant le paramètre `Protocol` pour spécifier le protocole `application`, l'élément `TopicArn` pour la rubrique à laquelle s'abonner ainsi que l'ARN d'un point de terminaison d'application mobile pour le paramètre `Endpoint`. Transmettez les paramètres à la méthode `subscribe` de la classe client `AWS.SNS`.

Pour appeler la `subscribe` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create subscribe/email parameters
var params = {
  Protocol: "application" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "MOBILE_ENDPOINT_ARN",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_subscribeapp.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `sns_subscribelambda.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant le paramètre `Protocol` pour spécifier le protocole `lambda`, l'élément `TopicArn` pour la rubrique à laquelle s'abonner ainsi que l'ARN d'une fonction AWS Lambda pour le paramètre `Endpoint`. Transmettez les paramètres à la méthode `subscribe` de la classe client `AWS.SNS`.

Pour appeler la `subscribe` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create subscribe/email parameters
var params = {
  Protocol: "lambda" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "LAMBDA_FUNCTION_ARN",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_subscribelambda.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `sns_unsubscribe.js`. Configurez le kit SDK comme illustré précédemment.

Créez un objet contenant le paramètre `SubscriptionArn`, en spécifiant l'ARN de l'abonnement à désabonner. Transmettez les paramètres à la méthode `unsubscribe` de la classe client `AWS.SNS`.

Pour appeler la `unsubscribe` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// 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 subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .unsubscribe({ SubscriptionArn: TOPIC_SUBSCRIPTION_ARN })
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_unsubscribe.js
```

Cet exemple de code se trouve ici sur 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 `AWS.SNS` client :

- `getSMSAttributes`
- `setSMSAttributes`
- `checkIfPhoneNumberIsOptedOut`
- `listPhoneNumbersOptedOut`
- `publish`

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier JSON d'informations d'identification, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

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 module Node.js nommé `sns_getsmstype.js`. Configurez le kit SDK comme illustré précédemment. 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 setSMSAttates 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 `setTopicAttributes` de la classe client `AWS.SNS`. Pour appeler la `getSMSAttributes` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create SMS Attribute parameter you want to get
var params = {
  attributes: [
    "DefaultSMSType",
    "ATTRIBUTE_NAME",
    /* more items */
  ],
};

// Create promise and SNS service object
var getSMSTypePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getSMSAttributes(params)
  .promise();

// Handle promise's fulfilled/rejected states
getSMSTypePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
```

```
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_getsmstype.js
```

Cet exemple de code se trouve ici sur GitHub.

Définition d'attributs SMS

Dans cet exemple, utilisez un module Node.js pour obtenir les attributs SMS actuels dans Amazon SNS. Créez un module Node.js nommé `sns_setsmstype.js`. Configurez le kit SDK comme illustré précédemment. 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 setSMSAttates 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 `setTopicAttributes` de la classe client `AWS.SNS`. Pour appeler la `getSMSAttributes` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create SMS Attribute parameters
var params = {
  attributes: {
    /* required */
    DefaultSMSType: "Transactional" /* highest reliability */,
    //'DefaultSMSType': 'Promotional' /* lowest cost */
  },
};

// Create promise and SNS service object
var setSMSTypePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .setSMSAttributes(params)
```

```
  .promise();

// Handle promise's fulfilled/rejected states
setSMSTypePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_setsmstype.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `sns_checkphoneoptout.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 `checkIfPhoneNumberIsOptedOut` de la classe client `AWS.SNS`. Pour appeler la `checkIfPhoneNumberIsOptedOut` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// 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 phonenumPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .checkIfPhoneNumberIsOptedOut({ phoneNumber: "PHONE_NUMBER" })
  .promise();

// Handle promise's fulfilled/rejected states
```

```
phonenumPromise
  .then(function (data) {
    console.log("Phone Opt Out is " + data.isOptedOut);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_checkphoneoptout.js
```

Cet exemple de code se trouve ici sur 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 module Node.js nommé `sns_listnumbersoptedout.js`. Configurez le kit SDK comme illustré précédemment. Créez un objet vide comme paramètre.

Transmettez l'objet à la méthode `listPhoneNumbersOptedOut` de la classe client `AWS.SNS`. Pour appeler la `listPhoneNumbersOptedOut` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// 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 phonelistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listPhoneNumbersOptedOut({})
  .promise();

// Handle promise's fulfilled/rejected states
phonelistPromise
  .then(function (data) {
    console.log(data);
  })
```

```
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_listnumbersoptedout.js
```

Cet exemple de code se trouve ici sur GitHub.

Publication d'un SMS

Dans cet exemple, utilisez un module Node.js pour envoyer un SMS à un numéro de téléphone. Créez un module Node.js nommé `sns_publishsms.js`. Configurez le kit SDK comme illustré précédemment. 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 `publish` de la classe client `AWS.SNS`. Pour appeler la `publish` méthode, créez une promesse pour appeler un objet de service Amazon SNS, en transmettant l'objet de paramètres. Traitez ensuite l'élément `response` dans le rappel de promesse.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create publish parameters
var params = {
  Message: "TEXT_MESSAGE" /* required */,
  PhoneNumber: "E.164_PHONE_NUMBER",
};

// Create promise and SNS service object
var publishTextPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .publish(params)
```

```
  .promise();

// Handle promise's fulfilled/rejected states
publishTextPromise
  .then(function (data) {
    console.log("MessageID is " + data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sns_publishsms.js
```

Cet exemple de code se trouve ici sur GitHub.

Exemples Amazon SQS

Amazon Simple Queue Service (Amazon SQS) est un service de file d'attente de messages rapide, fiable, évolutif et entièrement géré. Amazon SQS vous permet de dissocier les composants d'une application cloud. Amazon SQS inclut des files d'attente standard avec un débit et un at-least-once traitement élevés, ainsi que des files d'attente FIFO qui fournissent une livraison FIFO (premier entré, premier sorti) et un traitement en une seule fois.

L' JavaScript API d'Amazon SQS est exposée par le biais de la classe `AWS.SQS` client. Pour plus d'informations sur l'utilisation de la classe client Amazon SQS, consultez la référence `Class: AWS.SQS`de l'API.

Rubriques

Utilisation des files d'attente dans Amazon SQS

Cet exemple de code Node.js présente :

- Comment récupérer la liste de toutes vos files d'attentes de messages
- Comment récupérer l'URL d'une file d'attente en particulier
- Comment créer et supprimer des files d'attente

À propos de l'exemple

Dans cet exemple, une série de modules Node.js est utilisée pour utiliser des files d'attentes. Les modules Node.js utilisent le SDK pour permettre JavaScript aux files d'attente d'appeler les méthodes suivantes de la classe `AWS.SQS` client :

- `listQueues`
- `createQueue`
- `getQueueUrl`
- `deleteQueue`

Pour plus d'informations sur les messages Amazon SQS, consultez la section Comment fonctionnent les files d'attente dans le manuel Amazon Simple Queue Service Developer Guide.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Liste de vos files d'attente

Créez un module Node.js nommé `sqs_listqueues.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon SQS, créez un objet de `AWS.SQS` service. Créez un objet JSON contenant les paramètres nécessaires pour répertorier vos files d'attentes, qui est par défaut un objet vide. Appelez la méthode `listQueues` pour extraire la liste des files d'attente. Le rappel retourne les URL de toutes les files d'attente.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_listqueues.js
```

Cet exemple de code se trouve ici sur GitHub.

Création d'une file d'attente

Créez un module Node.js nommé `sqs_createqueue.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon SQS, créez un objet de `AWS.SQS` service. Créez un objet JSON contenant les paramètres obligatoires pour répertorier vos files d'attente, qui doit inclure le nom de la file d'attente créée. Les paramètres peuvent également contenir des attributs pour la file d'attente, comme le nombre de secondes de retard dans la remise d'un message ou le nombre de secondes pour conserver un message reçu. Appelez la méthode `createQueue`. Le rappel retourne l'URL de la file d'attente créée.

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

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_createqueue.js
```

Cet exemple de code se trouve ici sur GitHub.

Récupération de l'URL d'une file d'attente

Créez un module Node.js nommé `sqs_getqueueurl.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon SQS, créez un objet de `AWS.SQS` service. Créez un objet JSON contenant les paramètres obligatoires pour répertorier vos files d'attente, qui doit inclure le nom de la file d'attente dont vous souhaitez obtenir l'URL. Appelez la méthode `getQueueUrl`. Le rappel retourne l'URL de la file d'attente spécifiée.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_getqueueurl.js
```

Cet exemple de code se trouve ici sur GitHub.

Suppression d'une file d'attente

Créez un module Node.js nommé `sqs_deletequeue.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon SQS, créez un objet de `AWS.SQS` service. Créez un objet JSON contenant les paramètres nécessaires pour supprimer une file d'attente, qui inclut l'URL de la file d'attente à supprimer. Appelez la méthode `deleteQueue`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_deletequeue.js
```

Cet exemple de code se trouve ici sur GitHub.

Envoi et réception de messages dans Amazon SQS

Cet exemple de code Node.js présente :

- Comment envoyer des messages dans une file d'attente.
- Comment recevoir des messages dans une file d'attente.
- Comment supprimer des messages dans une file d'attente.

Scénario

Dans cet exemple, une série de modules Node.js est utilisée pour envoyer et recevoir des messages. Les modules Node.js utilisent le SDK pour JavaScript envoyer et recevoir des messages en utilisant les méthodes suivantes de la classe `AWS.SQS` client :

- `sendMessage`
- `receiveMessage`
- `deleteMessage`

Pour plus d'informations sur les messages Amazon SQS, consultez Envoyer un message à une file d'attente Amazon SQS et Recevoir et supprimer un message d'une file d'attente Amazon SQS dans le guide du développeur Amazon Simple Queue Service.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créer une file d'attente Amazon SQS. Pour obtenir un exemple de création de file d'attente, consultez Utilisation des files d'attente dans Amazon SQS.

Envoi d'un message à une file d'attente

Créez un module Node.js nommé `sqs_sendmessage.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon SQS, créez un objet de `AWS.SQS` service. Créez un objet JSON contenant les paramètres obligatoires pour votre message, y compris l'URL de la file d'attente à laquelle vous souhaitez envoyer ce message. Dans cet exemple, le message fournit des détails au sujet d'un livre se trouvant sur une liste des meilleurs ouvrages de fiction, y compris le titre, l'auteur, et le nombre de semaines sur la liste.

Appelez la méthode `sendMessage`. Le rappel retourne l'ID unique du message.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_sendmessage.js
```

Cet exemple de code se trouve ici sur GitHub.

Réception et suppression des messages provenant d'une file d'attente

Créez un module Node.js nommé `sqs_receivemessage.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon SQS, créez un objet de `AWS.SQS` service. Créez un objet JSON contenant les paramètres obligatoires pour votre message, y compris l'URL de la file d'attente à partir de laquelle vous souhaitez recevoir ce message. Dans cet exemple, les paramètres spécifient la réception de tous les attributs de message, ainsi que la réception de 10 messages maximum.

Appelez la méthode `receiveMessage`. Le rappel retourne un tableau d'objets `Message` à partir duquel vous pouvez récupérer la valeur `ReceiptHandle` pour chaque message que vous utilisez, afin de supprimer ultérieurement le message. Créez un autre objet JSON contenant les paramètres obligatoires pour supprimer le message, qui sont l'URL de la file d'attente et la valeur `ReceiptHandle`. Appelez la méthode `deleteMessage` pour supprimer le message reçu.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_receivemessage.js
```

Cet exemple de code se trouve ici sur GitHub.

Gestion du délai de visibilité dans Amazon SQS

Cet exemple de code Node.js présente :

- Comment spécifier l'intervalle de temps pendant lequel les messages reçus par une file d'attente ne sont pas visibles.

Scénario

Dans cet exemple, le module Node.js est utilisé pour gérer le délai de visibilité. Le module Node.js utilise le SDK pour gérer le délai JavaScript d'expiration de visibilité en utilisant cette méthode de la classe `AWS.SQS` client :

- `changeMessageVisibility`

Pour plus d'informations sur le délai de visibilité d'Amazon SQS, consultez Visibility Timeout dans le manuel Amazon Simple Queue Service Developer Guide.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créer une file d'attente Amazon SQS. Pour obtenir un exemple de création de file d'attente, consultez Utilisation des files d'attente dans Amazon SQS.
- Envoyez un message à la file d'attente. Pour obtenir un exemple d'envoi de message à une file d'attente, consultez Envoi et réception de messages dans Amazon SQS.

Modification du délai de visibilité

Créez un module Node.js nommé `sqs_changingvisibility.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon Simple Queue Service, créez un objet `AWS.SQS` de service. Recevoir le message de la file d'attente.

À la réception du message de la file d'attente, créez un objet JSON contenant les paramètres obligatoires pour définir le délai d'expiration, y compris l'URL de la file d'attente contenant le message, la valeur `ReceiptHandle` retournée lorsque le message a été reçu, et le nouveau délai d'expiration en secondes. Appelez la méthode `changeMessageVisibility`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_changingvisibility.js
```

Cet exemple de code se trouve ici sur GitHub.

Activation de l'attente active de longue durée dans Amazon SQS

Cet exemple de code Node.js présente :

- Comment activer l'attente active de longue durée pour une nouvelle file d'attente

- Comment activer l'attente active de longue durée pour une file d'attente existante
- Comment activer l'attente active de longue durée lors de la réception d'un message

Scénario

Les longues interrogations réduisent le nombre de réponses vides en permettant à Amazon SQS d'attendre pendant un certain temps qu'un message soit disponible dans la file d'attente avant d'envoyer une réponse. L'attente active de longue durée élimine également les fausses réponses vides en interrogeant tous les serveurs, à la place d'un simple échantillon. Pour activer l'attente active de longue durée, vous devez spécifier un temps d'attente différent de zéro pour les messages reçus. Pour ce faire, vous devez définir le paramètre `ReceiveMessageWaitTimeSeconds` d'une file d'attente ou le paramètre `WaitTimeSeconds` à la réception d'un message.

Dans cet exemple, une série de modules Node.js est utilisée pour activer l'attente active de longue durée. Les modules Node.js utilisent le SDK pour activer les longues JavaScript interrogations à l'aide des méthodes suivantes de la classe `AWS.SQS` client :

- `setQueueAttributes`
- `receiveMessage`
- `createQueue`

Pour plus d'informations sur les longs sondages Amazon SQS, consultez la section Long Polling du manuel Amazon Simple Queue Service Developer Guide.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.

Activation de l'attente active de longue durée lors de la création d'une file d'attente

Créez un module Node.js nommé `sqs_longpolling_createqueue.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon SQS, créez un objet de `AWS.SQS` service. Créez un objet JSON contenant les paramètres obligatoires pour créer une file d'attente, y compris une valeur différente de zéro pour le paramètre `ReceiveMessageWaitTimeSeconds`. Appelez la méthode `createQueue`. L'attente active de longue durée est activée pour la file d'attente.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_longpolling_createqueue.js
```

Cet exemple de code se trouve ici sur GitHub.

Activation de l'attente active de longue durée pour une file d'attente existante

Créez un module Node.js nommé `sqs_longpolling_existingqueue.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon Simple Queue Service, créez un objet `AWS.SQS` de service. Créez un objet JSON contenant les paramètres obligatoires

pour définir les attributs de la file d'attente, y compris une valeur différente de zéro pour le paramètre `ReceiveMessageWaitTimeSeconds` et l'URL de la file d'attente. Appelez la méthode `setQueueAttributes`. L'attente active de longue durée est activée pour la file d'attente.

```
// 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 = {
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.setQueueAttributes(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_longpolling_existingqueue.js
```

Cet exemple de code se trouve ici sur GitHub.

Activation de l'attente active de longue durée pour la réception des messages

Créez un module Node.js nommé `sqs_longpolling_receivemessage.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon Simple Queue Service, créez un objet `AWS.SQS` de service. Créez un objet JSON contenant les paramètres obligatoires pour recevoir des messages, y compris une valeur différente de zéro pour le paramètre `WaitTimeSeconds` et l'URL de la file d'attente. Appelez la méthode `receiveMessage`.

```
// 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 exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_longpolling_receivemessage.js
```

Cet exemple de code se trouve ici sur GitHub.

Utilisation des files d'attente de lettres mortes dans Amazon SQS

Cet exemple de code Node.js présente :

- Comment utiliser une file d'attente pour recevoir et mettre en attente des messages en provenance d'autres files d'attente que celles-ci ne peuvent pas traiter.

Scénario

Une file d'attente de lettres mortes peut être ciblée par d'autres files d'attente (source) pour les messages ne pouvant pas être traités avec succès. Vous pouvez mettre de côté et isoler ces messages dans la file d'attente de lettres mortes pour déterminer pourquoi leur traitement a échoué. Vous devez configurer individuellement chaque file d'attente source qui envoie des messages à une file d'attente de lettres mortes. Plusieurs files d'attente peuvent cibler une seule file d'attente de lettre morte.

Dans cet exemple, un module Node.js est utilisé pour acheminer des messages vers une file d'attente de lettres mortes. Le module Node.js utilise le SDK pour utiliser les files JavaScript d'attente en lettres mortes en utilisant cette méthode de la classe `AWS.SQS` client :

- `setQueueAttributes`

Pour plus d'informations sur les files d'attente d'Amazon SQS, consultez la section Utilisation des files d'attente d'Amazon SQS Dead Letter dans le guide du développeur Amazon Simple Queue Service.

Tâches prérequises

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

- Installez Node.js. Pour plus d'informations sur l'installation de Node.js, consultez le site web de Node.js.
- Créez un fichier de configurations partagé avec vos informations d'identification utilisateur. Pour plus d'informations sur le fichier d'informations d'identification partagé, consultez Chargement des informations d'identification dans Node.js à partir du fichier d'informations d'identification partagé.
- Créez une file d'attente Amazon SQS qui servira de file d'attente lettre morte. Pour obtenir un exemple de création de file d'attente, consultez Utilisation des files d'attente dans Amazon SQS.

Configuration de files d'attente source

Après avoir créé une file d'attente comme file d'attente de lettres mortes, vous devez configurer les autres files d'attente qui acheminent des messages non traités vers la file d'attente de lettres mortes.

Pour ce faire, spécifiez une stratégie de redirection identifiant la file d'attente à utiliser comme file d'attente de lettres mortes, et le nombre maximum de réceptions pour les messages individuels avant que ces derniers soient acheminés vers la file d'attente de lettres mortes.

Créez un module Node.js nommé `sqs_deadletterqueue.js`. Veillez à configurer le kit SDK comme indiqué précédemment. Pour accéder à Amazon SQS, créez un objet de `AWS.SQS` service. Créez un objet JSON contenant les paramètres obligatoires pour mettre à jour les attributs de file d'attente, en incluant le paramètre `RedrivePolicy` qui spécifie à la fois l'ARN de la file d'attente de lettres mortes et la valeur `maxReceiveCount`. Spécifiez également l'URL de file d'attente source que vous souhaitez configurer. Appelez la méthode `setQueueAttributes`.

```
// 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 = {
  Attributes: {
    RedrivePolicy:
      '{"deadLetterTargetArn":"DEAD_LETTER_QUEUE_ARN","maxReceiveCount":"10"}',
  },
  QueueUrl: "SOURCE_QUEUE_URL",
};

sqs.setQueueAttributes(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Pour exécuter l'exemple, entrez ce qui suit dans la ligne de commande.

```
node sqs_deadletterqueue.js
```

Cet exemple de code se trouve ici sur GitHub.

Didacticiels

Les didacticiels suivants vous montrent comment effectuer différentes tâches liées à l'utilisation d' AWS SDK for JavaScript.

Rubriques

- Tutoriel : Configuration de Node.js sur une instance Amazon EC2

Tutoriel : 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 suppose que vous avez déjà lancé une instance Linux avec un nom DNS public accessible depuis Internet et à laquelle vous pouvez vous connecter à l'aide de SSH. Pour plus d'informations, consultez Étape 1 : Lancer une instance dans le guide de l'utilisateur Amazon EC2.

> **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.

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 dans 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 utiliser nvm pour installer Node.js, car il 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 utiliser nvm pour activer la version du nœud installé. Si l'instance est interrompue, vous devez réinstaller le nœud. L'alternative consiste à 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 AMI (Amazon Machine Image)

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.

Ressources connexes

Pour plus d'informations sur les commandes et les logiciels utilisés dans cette rubrique, consultez les pages web suivantes :

- gestionnaire de version de nœud (nvm) : voir nvm repo on. GitHub
- gestionnaire de package de nœud (npm) : consultez site web npm.

JavaScript Référence d'API

Les rubriques de référence de l'API relatives à la dernière version du SDK pour se JavaScript trouvent à l'adresse suivante :

AWS SDK for JavaScript Guide de référence de l'API.

SDK Changelog activé GitHub

Le changelog pour les versions 2.4.8 et ultérieures est disponible à l'adresse :

Journal des modifications.

Sécurité de ce AWS produit ou service

Chez Amazon Web Services (AWS), la sécurité dans le cloud est la priorité principale. En tant que client AWS , vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des organisations les plus pointilleuses sur la sécurité. La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cela comme la sécurité du cloud et la sécurité dans le cloud.

Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute tous les services proposés dans le AWS cloud et de vous fournir des services que vous pouvez utiliser en toute sécurité. Notre responsabilité en matière de sécurité est notre priorité absolue AWS, et l'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des programmes de AWS conformité.

Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez et par d'autres facteurs, notamment la sensibilité de vos données, les exigences de votre organisation et les lois et réglementations applicables.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Rubriques

- Protection des données dans ce AWS produit ou service
- Gestion de l’identité et des accès
- Validation de conformité pour ce AWS produit ou service
- Résilience pour ce AWS produit ou service
- Sécurité de l'infrastructure pour ce AWS produit ou service
- Appliquer une version minimale de TLS

Protection des données dans ce AWS produit ou service

Le modèle de responsabilité AWS partagée de s'applique à la protection des données dans ce AWS produit ou service. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur

cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des AWS services que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez Questions fréquentes (FAQ) sur la confidentialité des données. Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog Modèle de responsabilité partagée 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 AWS services.
- 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 plus d'informations 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 AWS services à l'aide de la console, de l'API ou AWS des SDK. AWS CLI Toutes les données que vous entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Gestion de l’identité et des accès

AWS Identity and Access Management (IAM) est un outil AWS service 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 AWS service outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

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' AWS services 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 AWS service 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 AWS service 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 AWS service 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, consultez 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 AWS services 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 AWS services 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 AWS services 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, consultez 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 AWS services cas, vous pouvez associer une politique directement à une ressource (au lieu d'utiliser un rôle comme proxy).

Pour connaître la différence entre les rôles et les politiques basées sur les ressources pour l'accès entre comptes, consultez la section Accès aux ressources entre comptes dans IAM dans le guide de l'utilisateur IAM.

- Accès multiservices — Certains AWS services utilisent des fonctionnalités dans d'autres AWS services. 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, un rôle 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, vous pouvez effectuer une action qui initie une autre action dans un autre service. FAS utilise les autorisations du principal appelant et AWS service, associées AWS service 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 AWS services 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.
  - Rôle de 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 AWS service 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. AWS service 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 Vue d'ensemble 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. 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. AWS services

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 principals (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 Vue d'ensemble 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 fonctionnalité 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 en résultant 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 fonctionnalités d'une organisation, vous pouvez appliquer les politiques de contrôle des services (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 séance en résultant 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 annule 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 en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

Comment AWS services travailler avec IAM

Pour obtenir une vue d'ensemble du AWS services 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 AWS service 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

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 AWS services 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 politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS ressources

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (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 AWS services 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 connaître la différence entre l'utilisation de rôles et de politiques basées sur les ressources pour l'accès entre comptes, consultez la section Accès aux ressources entre comptes dans IAM dans le guide de l'utilisateur d'IAM.

Validation de conformité pour ce AWS produit ou service

Pour savoir si un programme AWS services de conformité AWS service s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez AWS services 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 AWS services 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 AWS services sont pas éligibles à la loi HIPAA. Pour plus d'informations, consultez le 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 AWS services 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 AWS service 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.
- Amazon GuardDuty — Cela AWS service détecte les menaces potentielles qui pèsent sur vos charges de travail Comptes AWS, vos conteneurs et vos données en surveillant votre environnement pour détecter toute activité suspecte et malveillante. GuardDuty peut vous aider à répondre à diverses exigences de conformité, telles que la norme PCI DSS, en répondant aux exigences de détection des intrusions imposées par certains cadres de conformité.
- AWS Audit Manager— Cela vous AWS service 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 minimale de TLS

> **Important**
>
> La AWS SDK for JavaScript v2 négocie automatiquement la version TLS de plus haut niveau prise en charge par un point de terminaison de AWS service donné. 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.

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.

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 Node Version Manager (nvm) en suivant les étapes décrites dans Installation et mise à jour de Node Version Manager. Ensuite, exécutez les commandes suivantes pour installer et utiliser une version spécifique de Node.js.

```
nvm install 11
nvm use 11
```

Enforcing 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.

```
const https = require("https");
const {NodeHttpHandler} = require("@aws-sdk/node-http-handler");
const {DynamoDBClient} = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_2_method'
            }
        )
    })
});
```

Enforcing 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.

```
const https = require("https");
const {NodeHttpHandler} = require("@aws-sdk/node-http-handler");
const {DynamoDBClient} = require("@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 permettant d'activer les protocoles SSL. Si vous utilisez Safari version 7 ou supérieure, le protocole TLS 1.2 est automatiquement activé.

Ressources supplémentaires

Les liens ci-après fournissent des ressources supplémentaires que vous pouvez utiliser avec le kit AWS SDK for JavaScript.

AWSManuel de référence des kits SDK et des outils

LeAWSManuel de référence des kits SDK et des outilscontient également des paramètres, des fonctionnalités et d'autres concepts fondamentaux communs à de nombreuxAWSKits SDK.

JavaScript Forum SDK

Vous pouvez trouver des questions et discussions sur des questions d'intérêt pour les utilisateurs du kit SDK pour les kits SDK pour JavaScript dans leJavaScript Forum SDK.

JavaScript Kit SDK et Manuel du développeur sur GitHub

Il existe plusieurs référentiels sur GitHub pour le kit SDK pour JavaScript.

- Le SDK actuel pour JavaScript est disponible dans la régionRéférentiels SDK.
- Le kit SDK pour JavaScript Manuel du développeur du kit (ce document) est disponible au format Markdown dans son propre kitréférentiel de documentation.
- Une partie de l'exemple de code inclus dans ce guide est disponible dans le référentiel d'exemple de code SDK.

JavaScript Kit SDK sur Gitter

Vous pouvez également trouver des questions et discussions sur le kit SDK pour JavaScript dans leJavaScript Communauté SDKsur Gitter.

Historique du document pour AWS SDK for JavaScript

- Version SDK : voir JavaScript Référence d'API
- Dernière mise à jour majeure de la documentation : 31 mars 2022

Historique du document

Le tableau ci-après décrit les modifications importantes dans chaque édition du AWS SDK for JavaScript après mai 2018. Pour recevoir les notifications concernant les mises à jour de cette documentation, abonnez-vous à un flux RSS.

Modification	Description	Date
Appliquer une version minimale du protocole TLS	Ajout d'informations sur TLS 1.3.	31 mars 2022
Affichage de photos dans un compartiment Amazon S3 à partir d'un navigateur	Ajout d'un exemple pour simplement afficher des photos dans des albums photos existants.	13 mai 2019
Configuration des informations d'identification dans Node.js, nouveaux choix de chargement des informations d'identification	Ajout d'informations à propos des informations d'identification qui sont chargées à partir du fournisseur d'informations d'identification ECS ou d'un processus d'informations d'identification configuré.	25 avril 2019
Informations d'identification utilisant un processus d'identification configuré	Ajout d'informations à propos des informations d'identification chargées à partir d'un processus d'informations d'identification configuré.	25 avril 2019
Nouveau Commencer à utiliser un script de navigateur	Getting Started in a Browser Script a été réécrit pour	14 juillet 2018

	simplifier l'exemple et pour accéder au service Amazon Polly pour envoyer du texte et renvoyer une synthèse vocale que vous pouvez lire dans le navigateur. Consultez Démarrage dans un script de navigateur pour connaître le nouveau contenu.	
Nouveaux exemples de code Amazon SNS	Quatre nouveaux exemples de code Node.js pour travailler avec Amazon SNS ont été ajoutés. Consultez les exemples de code Amazon SNS.	29 juin 2018
Nouveau Démarrage dans Node.js	La section « Démarrage dans Node.js » a été réécrite afin d'utiliser un exemple de code mis à jour et de fournir plus de détails sur la façon de créer le fichier `package.json` , ainsi que le code Node.js lui-même. Consultez Démarrage dans Node.js pour connaître le nouveau contenu.	4 juin 2018

Mises à jour antérieures

Le tableau ci-après décrit les modifications importantes apportées à chaque publication du AWS SDK for JavaScript avant le 27 juin 2018.

Modification	Description	Date
Nouveaux exemples de code AWS Elemental MediaConvert	Ajout de trois nouveaux exemples de code Node.js en	21 mai 2018

Modification	Description	Date
	vue d'une utilisation avec AWS Elemental MediaConvert. Pour obtenir un exemple de code, consultez AWS Elemental MediaConvertExemples.	
Nouvelle modification sur le GitHub bouton	L'en-tête de chaque sujet comporte désormais un bouton qui vous amène à la version annotée du même sujet, afin que vous GitHub puissiez apporter des modifications afin d'améliorer la précision et l'exhaustivité du guide.	21 février 2018
Nouvelle rubrique sur les points de terminaison personnalisés	Ajout d'informations sur le format et l'utilisation des points de terminaison personnal isés pour l'exécution d'appels d'API. Consultez Spécifica tion des points de terminaison personnalisés.	20 février 2018

Modification	Description	Date
Guide du SDK pour les JavaScript développeurs sur GitHub	Le SDK pour le guide JavaScript du développeur est disponible au format Markdown dans son propre référentiel de documentation. Vous pouvez publier les problèmes que vous souhaiteriez voir abordés dans le manuel ou envoyer des demandes d'extraction pour soumettre les modifications proposées.	16 février 2018
Nouvel exemple de code Amazon DynamoDB	Un nouvel exemple de code Node.js permettant de mettre à jour une table DynamoDB à l'aide du client de documents a été ajouté. Pour obtenir un exemple de code, consultez Utilisation du client de documents DynamoDB.	14 février 2018
Nouvelle rubrique sur AWS Cloud9	Une rubrique expliquant comment utiliser AWS Cloud9 pour développer et déboguer le code du navigateur et le code Node.js a été ajoutée. Consultez Utilisation d'AWS Cloud9 avec l'AWS SDK for JavaScript.	5 février 2018

Modification	Description	Date
Nouvelle rubrique sur la journalisation du kit SDK	Une rubrique décrivant comment enregistrer les appels d'API effectués avec le SDK pour JavaScript a été ajoutée, y compris des informations sur l'utilisation d'un enregistreur tiers. Consultez Journalisation des appels AWS SDK for JavaScript.	5 février 2018
Mise à jour de la rubrique sur la définition des régions	La rubrique expliquant comment définir la région utilisée avec le kit SDK a été mise à jour et développée. Elle inclut notamment des informations sur l'ordre de priorité pour la définition de la région. Consulter Configuration de la AWS région.	12 décembre 2017
Nouveaux exemples de code Amazon SES	La section contenant des exemples de code du SDK a été mise à jour pour inclure cinq nouveaux exemples d'utilisation avec Amazon SES. Pour plus d'informations sur ces exemples de code, consultez Exemples de services de messagerie Amazon Simple.	9 novembre 2017

Modification	Description	Date
Améliorations de la facilité d'utilisation	Suite à de récents tests d'exploitabilité, un certain nombre de modifications ont été apportées afin d'amélior er la facilité d'utilisation de la documentation. • Les exemples de code sont plus clairement identifiés comme destinés à l'exécuti on avec un navigateur ou avec Node.js. • Les liens de la table des matières ne sautent plus immédiatement vers d'autres contenus web, y compris la référence d'API. • Inclut des liens supplémen taires dans la section Getting Started vers des informations détaillées sur l'obtention des AWS informations d'identification. • La documentation fournit plus d'informations sur les fonctionnalités Node.js courantes nécessaires pour utiliser le kit SDK. Pour plus d'informations, consultez Considérations à propos de Node.js.	9 août 2017

Modification	Description	Date
Nouveaux exemples de code DynamoDB	La section contenant des exemples de code du SDK a été mise à jour pour réécrire les deux exemples précédents et ajouter trois nouveaux exemples d'utilisation de DynamoDB. Pour plus d'informations sur ces exemples de code, consultez Exemples Amazon DynamoDB.	21 juin 2017
Nouveaux exemples de code IAM	La section contenant des exemples de code du SDK a été mise à jour pour inclure cinq nouveaux exemples de travail avec IAM. Pour plus d'informations sur ces exemples de code, consultez AWSExemples IAM.	23 décembre 2016
Nouveaux exemples CloudWatch de code et d'Amazon SQS	La section contenant des exemples de code du SDK a été mise à jour pour inclure de nouveaux exemples d'utilisation avec CloudWatch et avec Amazon SQS. Pour plus d'informations sur ces exemples de code, consultez CloudWatch Exemples Amazon et Exemples Amazon SQS.	20 décembre 2016

Modification	Description	Date
Nouveaux exemples de code Amazon EC2	La section contenant des exemples de code du SDK a été mise à jour pour inclure cinq nouveaux exemples d'utilisation d'Amazon EC2. Pour plus d'informations sur ces exemples de code, consultez Exemples Amazon EC2.	le 15 décembre 2016
Liste des navigateurs pris en charge plus visible	La liste des navigateurs pris en charge par le SDK pour JavaScript, qui se trouvait précédemment dans la rubrique sur les prérequis , a été dotée de sa propre rubrique afin de la rendre plus visible dans la table des matières.	16 novembre 2016

Modification	Description	Date
Première publication du nouveau manuel du développeur	Le manuel du développeur précédent est désormais obsolète. Le nouveau manuel du développeur a été réorganisé afin que les informations soient plus faciles à trouver. Lorsque les JavaScript scénarios Node.js ou de navigateur présentent des considérations particulières, celles-ci sont identifiées comme appropriées. Ce manuel fournit également des exemples de code supplémentaires qui sont mieux organisés afin d'être plus faciles à trouver.	28 octobre 2016